Abstracting User Metrics for Intelligent User Interfaces in MDD: A UsiXML Case Study

Alberto Gaspar[1][0000−0002−9236−4391], José Ignacio Panach[1][0000−0002−7043−6227], Miriam Gil[1][0000−0002−2987−1825], and Jean Vanderdonckt[2][0000−0003−3275−3333]

[1] Universitat de València, Av. de la universitat s/n, Burjassot, 46100, Valencia, Spain. algasvi@alumni.uv.es {joigpana,miriam.gil}@uv.es
[2] Université catholique de Louvain, Louvain School of Management, Place des Doyens, 1, Louvain-la-Neuve, 1348, Belgium.
jean.vanderdonckt@uclouvain.be

Abstract. Intelligent User Interfaces (IUIs) rely on user metrics to infer user characteristics and adapt interaction accordingly, yet these metrics are often specified in an implementation-dependent manner and become tightly coupled to a particular application context and technology stack. This paper proposes a model-driven abstraction process to specify user metrics independently of the underlying programming language, enabling their integration into Model-Driven Development (MDD) workflows. Building on a curated set of automatic user metrics for knowledge, skills, and goals, we describe how metric parameters can be expressed using conceptual user interface primitives and context constructs. We instantiate the approach with UsiXML and illustrate, through a practical scenario, how abstract metric values can drive systematic UI adaptations at the model level. To evaluate the feasibility of the proposal, we conducted a preliminary expert-based validation with three MDD specialists who evaluated the operationalization of the proposed metrics in UsiXML, WebRatio, and Integranova languages. Results indicate that most metrics can be supported across tools, while time-dependent metrics reveal limitations in environments lacking explicit temporal primitives.

Keywords: Model-Driven Development · Intelligent User Interfaces · User Metrics · UsiXML · User characteristics.

1 Introduction

Intelligent User Interfaces (IUIs) aim to tailor interaction to individual users by adapting content, presentation, and interaction mechanisms based on evidence collected during use [5]. A common way to operationalize such adaptation is through user metrics derived from interaction traces (e.g., time, clicks, scrolling, navigation patterns), which are subsequently mapped to higher-level user characteristics (UCs) such as knowledge, skills, preferences, or goals. This metric-driven view is attractive because it enables non-intrusive personalization and can be deployed continuously at runtime without requiring explicit questionnaires or

manual profiling steps. Despite extensive work on user modeling and adaptive interfaces, however, most existing IUI solutions remain highly specialized and tightly coupled to a single application context and technology stack, which increases development complexity and limits reuse. For example, Williford et al. [24] implemented a system capable of recognizing users' rectilinear strokes to support real-time freehand perspective drawing, a solution specifically tailored to a particular interaction scenario.

A key reason for this specialization is that the specification and reuse of user metrics remains a recurrent challenge. In practice, metrics are frequently defined at the implementation level—embedded in application code, bound to specific UI widgets or event handlers, and dependent on platform-specific interaction mechanisms (e.g., DOM events, mobile gestures) [1]. As a consequence, metric definitions become hard to port across applications, difficult to maintain as the UI evolves, and costly to replicate when moving to a different technology stack.

This situation is particularly problematic in Model-Driven Development (MDD) settings. MDD addresses software complexity by shifting effort toward conceptual models that abstractly represent system aspects and enable their transformation into executable code. By separating the problem space from the solution space, MDD promotes abstraction, systematic reuse, and consistent generation across platforms. While MDD frameworks offer structured representations of tasks, user interfaces, and context, the instrumentation required to compute metrics for IUIs is often reintroduced at low level, undermining the benefits of model-driven engineering. Moreover, existing approaches for supporting IUIs at the conceptual-model level provide limited guidance on how to specify and operationalize adaptation based on UCs within the problem space.

While interface adaptation has been extensively studied in Human–Computer Interaction at the implementation level [22], there are fewer contributions addressing how to manage metric-driven adaptation directly within conceptual models. Some proposals incorporate adaptation capabilities into MDD-based UI generation, but they often rely on predefined rules rather than dynamic, user-driven adaptation [13]. Likewise, multilayer UI modeling frameworks such as *MARIA* [16] provide structured representations of the UI, yet they do not explicitly integrate reusable mechanisms to define interaction-based metrics and connect them to runtime adaptation decisions. As a result, there remains a gap in systematically incorporating metric-driven UI adaptation within conceptual models, bridging MDD and adaptive user interfaces.

This paper addresses this gap by proposing a model-driven abstraction process for user metrics, designed to reduce coupling between metric definitions and concrete implementations. These metrics are used to determine each UC during interaction. Our approach abstracts these metrics, making them adaptable to any MDD method that supports interface representation through models. Our approach builds on a curated set of automatic user metrics for characterizing user knowledge, skills, and goals (as defined in prior work [8]) and focuses on making those metrics reusable across applications and platforms.

We instantiate the process using UsiXML [12], a well-established model-based UI notation, and illustrate its applicability through a practical scenario involving user interaction with a stock-selection interface. The scenario shows how metric parameters can be mapped to UsiXML primitives and how metric values can drive systematic adaptation rules at the model level (e.g., emphasizing relevant options, or simplifying choices). Furthermore, these instantiations have important implications for MDD, as incorporating user preference data into the development life cycle can lead to more user-centered design processes [4]. By explicitly integrating user preference metrics into UI models, developers can ensure that the resulting systems are both usable and appealing to their target users, thereby aligning subjective satisfaction with objective performance outcomes. To assess the feasibility of the proposal, we complement the UsiXML instantiation with a preliminary expert-based validation involving three MDD specialists who analyzed whether the proposed metrics can be operationalized in UsiXML, WebRatio, and Integranova. The feedback highlights broad feasibility for most metrics and identifies limitations for time-dependent constructs in environments lacking explicit temporal primitives.

Specifically, the contributions of the paper are: (1) A model-driven abstraction process to specify user metrics independently of programming language and platform, enabling reuse in MDD workflows, and (2) An operational mapping of a set of user metrics to model-level UI primitives, instantiated in UsiXML and exemplified through a realistic interaction scenario.

This work is structured as follows. Section 2 reviews the related work. Section 3 presents the user metrics, the UsiXML models, and the proposed abstraction process. Section 4 applies this process to a practical example. Section 5 shows a preliminary validation of the process. Finally, Section 6 concludes the paper and discusses directions for future research.

2 Related work

The abstraction of user metrics for extracting user characteristics has been widely discussed in the literature. Prior work focuses on organizing and structuring metric components to enable their extraction. This abstraction is necessary to ensure that user metrics can be consistently extracted across different programming languages used to develop the system.

This group includes works that propose Domain-Specific Languages (DSL) for the declarative definition of user metrics. Such DSLs allow metrics to be expressed at a high level of abstraction, focusing on domain concepts rather than implementation details. These approaches aim to improve the expressiveness, reuse, and maintainability of metrics, while facilitating their adaptation across different programming languages and execution environments. MODNESS [6] is an MDD approach that enables the high-level specification and automated assessment of fairness in software systems through a dedicated domain-specific language. The approach supports the definition and composition of fairness metrics independently of the application domain and provides automated code gen-

eration for metric evaluation. Its effectiveness is demonstrated through multiple cross-domain use cases, showing improvements in expressiveness and automation over existing fairness assessment solutions. L-PRISM [17] is a domain-specific language supported by a dedicated metamodel for modeling multimedia service function chains in edge–cloud environments. Following an MDD approach, the language enables the declarative specification of domain concepts while abstracting away low-level deployment and configuration details. The authors validate L-PRISM through a proof-of-concept and developer-based experiments, demonstrating improvements in expressiveness, usability, maintainability, and reduced configuration complexity compared to manual approaches. Dalla Palma et al. [15] introduce a catalogue of domain-specific metrics for assessing the quality of infrastructure-as-code scripts, emphasizing the importance of defining metrics aligned with domain constructs rather than general-purpose programming elements. Their approach is validated through real-world case studies that demonstrate how DSL-level metrics can effectively support quality assessment and system evolution. While this work highlights the advantages of domain-specific and model-driven metric definitions at a high level of abstraction, it does not address user-oriented metrics in the context of IUI. In this work, we bridge this gap by combining MDD with IUI design to declaratively specify user metrics.

Some proposals use models to describe user behaviors when the user interacts with the system. Sharma et al. [20] propose a set of user interaction metrics capturing session duration, performed activities, device characteristics, and document downloads, which are integrated into an unsupervised model. The approach is evaluated across three goal-oriented interaction scenarios, achieving 90.17% accuracy and 91.07% true positive rate. While effective for user behavior analysis, the work does not address the declarative specification of user metrics or their integration into model-driven IUI design. The Deep Interest Highlight Network (DIHN) [21] evaluates the advantages of Trigger-Induced Recommendation (TIR) systems over non-TIR systems by modeling users' immediate interests for Click-Through Rate prediction. The model analyzes user interactions and clicks to assess intention levels, tracking accessed options and performed actions to identify dynamic user interests. A/B test results indicate that DIHN improves accuracy by 6% compared to non-TIR-based systems. Guo et al. [10] propose a user behavior model for predicting system trust levels aimed at improving user satisfaction and engagement. The model integrates user metrics such as concurrent connections, session duration, and user actions, and is validated through simulation-based experiments. The results demonstrate the model's scalability and effectiveness in supporting engagement-oriented recommendations. The main limitation of these works is that the proposed metrics are not user-centred, and all validation phases are conducted in simulation without real users. This means that the presented metrics may not be representative.

IUIs aim to analyse users' actions and behaviours to dynamically adapt the graphical user interface to the user. Chart4Blind [14] is a system aimed at improving chart accessibility for impaired users by analyzing interaction traces with visual chart elements. Based on observed user behavior, the system auto-

matically generates accessible representations, tactile graphics and textual descriptions. The approach is validated through usability studies with both sighted and visually impaired participants, demonstrating its effectiveness in enhancing accessibility. FootApp [2] is an IUI for football match annotation. It integrates multimodal interaction, with sensor-based activity recognition to facilitate annotation tasks. The system leverages interaction data and wearable sensors to build user models and infer implicit user information. Its effectiveness is demonstrated through real-world match experiments, highlighting its practical applicability. CARSI 3.0 [23] stores driving context and historical preferences to recommend infotainment actions that reduce driver distraction and improve usability. User behavior is modeled by combining interaction history with static and dynamic contextual data and in-vehicle sensor information. The approach is validated on a large-scale real-world dataset collected from multiple vehicles, demonstrating effective context-aware interface adaptation. The main limitation of these works is that the IUI proposed are highly specialized to specific application contexts, and the associated user models are tightly coupled to domain-specific assumptions. This implies that the presented approaches cannot be easily generalized or reused across different systems or development environments.

Our work builds on previous findings by combining concepts from IUIs and MDD to define a set of UCs that can be abstracted from system-specific details and applied across a wide range of contexts. Using existing metrics from the literature as a foundation, we define the parameters required for their calculation, ensuring that user metrics can be computed through a clear and reusable process.

3 Applying user metrics of IUIs to MDD

This section introduces the user metrics considered in the study and how they are abstracted for use in MDD. The metrics are derived from prior research and are designed to be independent of programming languages and platforms. We then provide a brief overview of UsiXML, which we use as a representative model-based UI notation to exemplify how the proposed abstract metrics can be instantiated in a concrete MDD approach.

3.1 User metrics

This subsection describes the user metrics employed by the system to extract user **skills, knowledge, and goal** UCs. These metrics were previously defined in [8]. Among all the available metrics, we focus on this subset because, as shown in previous work [8], they can be automatically captured during user–system interaction in a non-intrusive manner. This characteristic makes them a key factor in the adaptability process. Table 1 summarizes the user metrics used to estimate each UC. Next, we describe each metric, its justification in the literature with a reference, and its utility to design IUIs.

The **Mc** metric aims to determine if the user makes the correct clicks to complete the task. It is calculated as the absolute difference between the parameter U_{clicks}, which represents the number of user clicks, and the estimated clicks

required to complete the task, Uc_{es} [27]. This estimation is done by an expert in graphical user interface design.

The **Msc** metric aims to ascertain whether the user performs the correct scrolling to complete the task. This metric is calculated as the absolute difference between the user's scroll (Scr_{user}) and the estimated scroll parameter (Scr_{es}) [3].

The **Sn** metric aims to determine how many times the user clicks the search option. This metric is calculated using a parameter that indicates the number of clicks the user performs on the search option [27].

The **A** metric aims to quantify the user's interaction accuracy by capturing deviations from the estimated optimal number of actions required to complete the task. This metric is computed as one minus the normalized absolute difference between the number of unique clicks performed by the user (Uc_{user}) and the estimated number of clicks required to complete the task (Uc_{es}) [26].

The **Tt** metric aims to determine if the user requires more time than expected to complete the task. This metric is calculated as the absolute difference between the estimated time parameter ($Time_{es}$) and the user's time required to complete the task ($Time_{user}$) [26].

The **Tp** metric aims to determine the average time per option that the user takes to complete the task. This metric is calculated as the division of the parameter measuring the total time of the user ($Time_{user}$) by the parameter measuring the number of options accessed ($pages_{accessed}$) [26].

The **Sp** metric aims to determine the sensible pauses that the user has made. A sensible pause is defined as a period of inactivity longer than 5 seconds. The metric is calculated using the parameter that measures the total duration of all pauses lasting more than 5 seconds [18].

The **Com** metric aims to determine if the user completes the task correctly. This metric is based on the percentage of tasks successfully completed [18]. For that, the system captures both the expected sequence of user actions for task completion and the actions actually executed by the user.

The **Qc** metric aims to determine the complexity of the queries needed to complete the task. It is based on several parameters: (1) The number of terms used; (2) the proportion of unique terms; (3) whether the user includes special characters to filter the products (such as " ", AND or OR); (4) whether the user accesses advanced search, and (5) the number of stop words used by the user. Stop words consist of common words that do not provide significant information for the search, such as prepositions or connectors [3].

The **Co** metric aims to determine if the user accesses more options than the required number. It is calculated as the absolute difference between the parameter measuring the options accessed by the user (Cuo) and the parameter measuring the different options estimated to complete the task ($options_{es}$) [3].

The **Spa** metric aims to determine the level of engagement with special actions performed by the user. A special action is a highly skilled action performed by the user, such as using keyboard shortcuts or advanced search menus to accomplish a task. To identify these actions, it is necessary to define the complete

set of available special actions the user can perform and to calculate the average number of these actions the user performs [19].

The **R** metric aims to determine the number of options the user revisits. An option is revisited when a user revisits a previously accessed option, such as when they cancel the purchase process while entering shipment data because they need to add another product to the cart. It is calculated as the absolute difference between the parameter that measures the options accessed by the user (Uo) and the parameter that measures the estimated options to complete the task ($options_{es}$) [19].

The **Opq** metric aims to determine the number of queries made by the user per session or interaction. It is calculated as the ratio of the parameter measuring the options the user interacts with (Ov) to the parameter measuring the number of searches the user makes in a session or interaction ($Queries$) [3].

The $\boldsymbol{T_{Mpag}}$ metric aims to determine the maximum time the user spends interacting with the most accessed option in the system. It is calculated by identifying the option the user interacts with for the longest duration. This is done by implementing a timer for each option when the user interacts with it, and then storing this duration. The system then determines the maximum time the user has spent on any given option [26].

Table 1. Overview of User Metrics and their User Characteristics

User Metric	**UCs Calculated**
Mouse click (MC) = $\lvert U_{clicks} - Uc_{es}\rvert$	knowledge, skills, goals.
Mouse scrolling (Msc) = $\lvert Scr_{user} - Scr_{es}\rvert$	knowledge.
Search number (Sn) = $\sum$(clicks on search)	knowledge, goals
Accuracy (A) = 1- $\frac{\lvert Uc_{user} - Uc_{es}\rvert}{Uc_{es}}$	knowledge, skills
Total time (Tt) = $\lvert Time_{user} - Time_{es}\rvert$	knowledge, skills
Time per page (Tp) = $\frac{Time_{total}}{pages_{accessed}}$	knowledge
Sensible pauses (Sp) = $\sum$(Inactivity $> 5s$)	knowledge
Complete ratio (Com) = Task completed?	knowledge, skills, goals
Query complexity (Qc) = Query complexity	knowledge, goals
Click options (Co) = $\lvert Cuo - options_{es}\rvert$	skills
Special actions (Spa) = $AVG(\#Sp_{act})$	skills
Revisited ratio (R) = $\lvert Dov - Ov\rvert$	skills
Options per query (Opq) = $\frac{Ov}{Queries}$	goals
Time max per page (T_{Mpag}) max($time\ pages$)	goals

3.2 UsiXML models

Metrics and their parameters are defined so they can be represented in conceptual models that abstractly model UIs, in accordance with the MDD paradigm. This facilitates their inclusion in existing model-to-model transformations and

model-to-code transformations. As an illustrative example, we use the MBUI - Abstract User Interface Models [25], which is the standard method for defining user interfaces abstractly. Specifically, the example uses UsiXML [12], a specific notation based on the standard, to demonstrate how metrics can be operationalized within these models.

Table 2. Classes used from the concrete UsiXML model and their representation

Model	**UsiXML Class**	**Representation**
Concrete	*ListBox*	Represents an element of a list.
Concrete	*CheckItem*	Represents a checkbox element.
Concrete	*MenuItem*	Represents an element of a menu.
Concrete	*ImageComponent*	Represents the icons of the UI.
Concrete	*InputText*	Represents a text input by the user.
Concrete	*Scroll*	Represents the different scroll performed by the user.
Concrete	*Button*	Represents a component that users click to interact with, typically to trigger specific actions within the interface.
Concrete	*Event*	Represents an expression to evaluate a specific action.
Context	*Temporalization*	Represents a timer.
Context	*Platform*	Represents a specific device.

UsiXML is an XML-based modeling language used to define a system's UIs. It is based on conceptual models that abstractly represent the UI across different abstraction layers. UsiXML is an MDD method that aims to develop systems only using conceptual models. UIs are automatically generated through model-to-model transformations and model-to-code transformations. The main advantage of this language is that the designed UIs are independent of the system's technology [12]. UsiXML provides a wide range of models for describing UIs, including the abstract model, concrete model, and task domain models. We focus on the classes specified in the UsiXML concrete and context models, as both contain the elements used as parameters for the metrics in this example. The concrete model describes how the different UI components are displayed and how they interact within a specific system. This model represents a layer of abstraction closer to the system's final implementation. Each class in this concrete model contains a label named *value* that identifies individual UI components. The context model describes which states of the entities may affect the system. Table 2 summarizes the classes of the concrete model and the context model used to calculate the user metrics, the primitives used, and the instantiation of those primitives in real widgets.

3.3 User metrics application to UsiXML

Next, we specify how the metrics defined in Table 1 can be calculated using UsiXML conceptual primitives. The following list presents each metric along with a description of how each metric is computed, indicating in parentheses the UsiXML conceptual primitives involved in its estimation.

- **Mouse click (Mc)**. The U_{clicks} parameter is computed as the sum of each click of the user in any primitive (*ListBox*, *CheckBox*, *MenuItem*, *ImageComponent*).
- **Mouse scrolling (**Msc**)**. The Scr_{user} parameter is computed with a scroll instance that evaluates the user's scroll action, and an event instance processes it. Then the system estimates the difference between these parameter and the Scr_{es} parameter (*Scroll*, *Event*).
- **Search Number (Sn)**. It is computed by the sum of the clicks performed by the user in the primitive marked as glass search icon (*ImageComponent*).
- **Accuracy (A).** The Uc_{user} parameter is computed by an *Event* instance, which evaluates whether each user click corresponds to a specific action or constitutes a misclick. (*CheckItem*, *Event*).
- **Total time (Tt).** The $Time_{user}$ parameter is computed by a *Temporalization* instance, which tracks time by starting a counter when the user begins interacting and stopping it when the user ends the task (*Temporalization*).
- **Time per page (Tp).** The parameter $Time_{total}$ is computed by an *Event* instance, which starts a counter when the user accesses a system option and stops it when the user accesses another option (*Temporalization*, *Event*).
- **Sensible pauses (Sp).** It is computed by a *Temporalization* instance, which initiates a counter at the user's last click and stops it upon the next click. An *Event* instance then determines whether the elapsed time exceeds five seconds (*Temporalization*, *Event*).
- **Complete ratio (Com).** It is computed by an *Event* instance, which checks whether the items pressed by the user are required to complete the task and estimates the average of correct actions (*CheckItem*, *Event*).
- **Query complexity (Qc)**. It is computed by an *Event* instance, which evaluates the text entered by the user in the *InputText* instance of the GUI (*InputText*, *Event*).
- **Click options (Co)**. The Cuo parameter is computed by an *Event* instance, which evaluates each user click and counts those that occur on elements marked as options (*CheckItem*, *ImageComponent*, *Button*, *Event*).
- **Special actions (Spa)**. It is computed as an *Event* instance, which evaluates user clicks on *ImageComponent* and *MenuItem* widgets and calculates the average number of special actions performed by the user (*Event*, *ImageComponent*, *MenuItem*).
- **Revisited ratio (R)**. The *Dov* parameter is computed with an *Event* instance, which evaluates whether the user's clicks occur on a previously defined option by evaluating the *CheckItem* and *ImageComponent* elements (*CheckItem*, *ImageComponent*, *Event*).
- **Options per query (Opq)**. The *ov* parameter is computed by an *Event* instance, which obtains the number of clicks in each option, and another event that calculates the total number of users' clicks (the value of the *Sn* metric) (*ImageComponent*, *Event*).
- **Time Max Per Page (**T_{Mpag}**)**. It is computed by a *Temporalization* instance, which measures the time the user spends on each page (like *Tp* user metric), and a separate instance that determines which time value is the highest (*Event*, *Temporalization*).

4 Practical case study

This section illustrates a practical application of the proposed metrics to UsiXML. In this case study, we demonstrate how interaction events represented at the model level can be used to compute metrics and drive UI adaptations.

4.1 Scenario and baseline GUI

We consider a stock-management interface of a home automation system. The user's task is to select all vegetable products and generate a shopping list. The goal is intentionally simple so that adaptations can be explained in terms of model-level primitives rather than domain logic. The interaction trace produced while completing the task is used to compute a set of metrics that support the inference of user characteristics (UCs) and the application of adaptation rules.

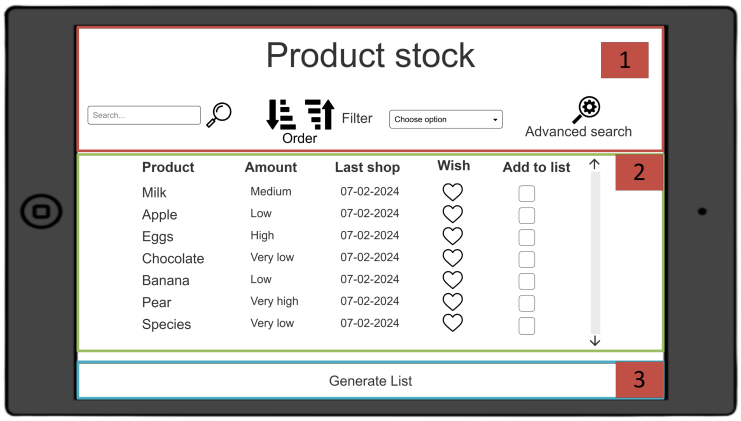

Fig. 1. Representation of the first UI example.

Figure 1 depicts the baseline GUI, represented textually as a UsiXML concrete UI model. The interface is organized into three functional sections:

- **Section 1 — Search and advanced actions.** This section contains icons for advanced search-related actions. These icons are instances of the *ImageComponent* class of the concrete model of UsiXML and have internal values that identify whether the user uses them or not (*search_icon*, *order*, *filter*, and *advanced_search*). Additionally, this section displays the search bar; which is an instance of *InputText* class with the value *search_bar*.
- **Section 2 - Product selection controls.** This section contains the options that the user can use to mark the product as a favourite; this element belongs to the *ImageComponent* class with the value *wish*. The checkbox that the user can click to add products to the cart belongs to the *ImageComponent* class and has the value *add_to_cart*.

– **Section 3 - Task completion control.** This section contains the button that allows the user to generate a shopping list with the selected products. This element belongs to the *Button* class with the value *generate_list*.

This baseline GUI is used as the reference point for the adaptations triggered by the metrics described below.

4.2 Computing user metrics from UsiXML primitives

Next, we describe how each user metric is estimated based on user interactions represented using the UsiXML language, and we define how the GUI primitives are customized according to the value of each metric, indicating which metrics affect each adaptation. For this purpose, we follow the usability guidelines presented in [9, 7].

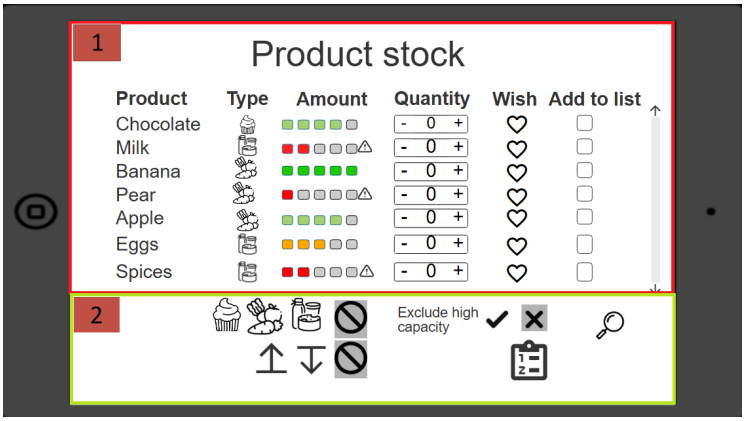

Fig. 2. Representation of the customized GUI example.

Figure 2 presents an example of a customized GUI for a user who has high knowledge and skills, and a transactional goal. The adaptation illustrates the kind of changes enabled by metric-driven adaptation at the model level. In this variant, the GUI provides additional information through icons, such as the type of product or filter options. Next, we describe how the values calculated with the metrics lead to such adaptations.

The **Skills** characteristic is computed using the following metrics: mouse clicks (Mc), mouse scrolling (Msc), special actions (Spa), and completion ratio (Com). In the following, we describe the calculation of each metric and its impact on the user interface.

The *Mc* metric is calculated as the difference between the estimated number of clicks (5) and the actual number of clicks performed by the user (5), resulting in a value of 0. The *Msc* metric is obtained as the difference between the

user's vertical scrolling distance (120) and the expected scrolling distance (100), yielding a value of 20. The *Spa* metric represents the average number of special actions performed per task. In this case, since only one task was defined and the user executed three special actions (product search, exclusion of unnecessary items, and filtering), the value of this metric is 3. Finally, the *Com* metric corresponds to the percentage of the task that was correctly completed by the user. As the user selected all three required products, the completion ratio is 100%.

Once all metrics have been computed, the overall value of the Skills characteristic is obtained as the average of these metrics. In this case, the resulting value is classified as *high*. According to usability guidelines [9, 7], a user with high skills should have flexibility in accessing the functionality. So, in *Section 1*, the GUI includes new options to complete the list such as the label indicating the amount of product and an input field to allow the user to specify the quantity of product purchased. In *Section 2*, the high value of the Skills characteristic triggers the inclusion of icons in the GUI that enable advanced search, filtering, ordering, and product exclusion operations. This adaptation aims to reduce scrolling activity and overall task completion time by providing direct access to task finalization once all required actions have been successfully completed.

The **Goals** characteristic is computed using the following metrics: search number (Sn), total time (Tt), revisited ratio (R), options per query (Opq), and maximum time per page ($T_M pag$). In the following, we describe the computation of each metric and its impact on the user interface.

The *Sn* metric is defined as the total number of searches performed by the user. In this case, the user carried out 3 searches, corresponding to the queries submitted via the magnifying glass icon. The *Tt* metric represents the total time required to complete the task. The observed task completion time was 50 seconds, compared to an expected time of 55 seconds. The *R* metric is calculated as the number of options revisited during the interaction. In this scenario, its value is 0, as the user did not reselect any previously chosen option. The *Opq* metric is computed as the ratio between the number of visited options (3) and the total number of queries issued by the user (3), resulting in a value of 1 option per query. Finally, the $T_M pag$ metric corresponds to the maximum amount of time the user spent on a single page, which in this case was 9 seconds.

Once all metrics have been computed, the overall value of the Goals characteristic is obtained as the average of these metrics. In this case, the resulting value is classified as *transactional*. According to [9, 7], users seeking comprehensive information are more likely to access it directly than to navigate through the system's various options. So, the transactional value of Goals leads to specific adaptations in the GUI: In Section 1, an information icon is added to display product quantity details, along with an alert indicating products with low availability. In Section 2, these metric values motivate a GUI adaptation in which only the magnifying glass icon is displayed by default, while the search bar is revealed dynamically when the user interacts with it.

The **Knowledge** characteristic is computed using the following metrics: accuracy (A), sensible pauses (Sp), query complexity (Qc), and click options (Co). The following paragraphs describe the computation of each metric and its impact on the user interface.

The *A* metric is defined as the percentage of actions correctly performed by the user. In this case, the accuracy reached 90%, considering the total number of interactions required to successfully complete the task. The *Sp* metric corresponds to the number of significant pauses detected during the user interaction. In this scenario, its value is 0, as no prolonged interruptions were observed throughout task execution. The *Qc* metric reflects the level of complexity of the queries issued by the user. In this case, the complexity is classified as high due to the combined use of search, filtering, and product ordering actions. The *Co* metric is calculated as the total number of options selected via mouse clicks. Here, the user selected three options, corresponding to the required actions to complete the task.

Once all metrics have been computed, the overall value of the Knowledge characteristic is obtained as the average of these metrics. In this case, the resulting value is classified as *expert*. According to [9, 7], this means the user knows how to do anything without any help. In Section 1, the GUI uses icons to represent product types, enabling faster visual recognition and reducing the need for additional user interaction. In Section 2, the textual search field is removed, as the user demonstrates sufficient expertise to efficiently utilize the remaining system functionalities.

5 Preliminary validation

Next, we show a preliminary validation of this work. For that, we interviewed 3 MDD experts to determine if the proposed metrics can be used in three different MDD methods: UsiXML [12], WebRatio [4] (a method for generating web and mobile applications and Integranova [11](a method for generating web and desktop applications). The experts had the following backgrounds: 2 years of experience with UsiXML, 20 years with Integranova, and 10 years with WebRatio, respectively. All of them are researchers in MDD and HCI.

A semi-structured interview approach was adopted as the primary data collection instrument. During these interviews, each user metric parameter was presented and discussed individually. Experts were asked to assess whether the information required by each metric could be derived from existing MDD languages, as well as to identify which specific language elements could support its implementation. When experts determined that a metric parameter could not be estimated using the available modeling constructs, the corresponding metric was considered as not supported.

The results showed in Table 3 indicate that most of the proposed metrics can be operationalized in the three analyzed MDD methods. However, metrics that rely on time-dependent parameters cannot be supported in Integranova. This is due to the absence of explicit temporal modeling primitives in Integranova,

which prevents the representation and computation of time-related information, in contrast to methods such as UsiXML and WebRatio that provide dedicated constructs for this purpose.

Table 3. Compliance of user metrics in MDD methods

User Metric	**UsiXML**	**Integranova**	**WebRatio**
Mouse click (MC)	Supported	Supported	Supported
Mouse scrolling (Scr_{user})	Supported	Supported	Supported
Search number (Sn)	Supported	Supported	Supported
Accuracy (A)	Supported	Supported	Supported
Total time (Tt)	Supported	Not Supported	Supported
Time per page (Tp)	Supported	Not Supported	Supported
Sensible pauses (Sp)	Supported	Not Supported	Supported
Complete ratio (Com)	Supported	Supported	Supported
Query complexity (Qc)	Supported	Supported	Supported
Click options (Co)	Supported	Supported	Supported
Special actions (Spa)	Supported	Supported	Supported
Revisited ratio (R)	Supported	Supported	Supported
Options per query (Opq)	Supported	Supported	Supported
Time max per page (T_{Mpag})	Supported	Not Supported	Supported

6 Conclusion and future works

We propose an abstraction of user metrics for integration into MDD methods and instantiate it in UsiXML. A preliminary expert assessment across UsiXML, Integranova, and WebRatio suggests that most metrics are broadly applicable, although Integranova exhibits the most limitations. This work is limited to metrics targeting skills, knowledge, and goals, and excludes metrics that require explicit user input (e.g., age). In addition, despite their intended system independence, deploying the metrics in concrete applications may still require extra mappings to platform-specific interaction mechanisms.

Despite these limitations, the approach offers clear benefits. First, abstracting user metrics from platform- and system-specific details supports reuse across multiple MDD languages and software environments. Second, the metrics are specified with sufficient precision while remaining suitably abstract, facilitating integration into diverse development methodologies. Third, the approach is applicable across different user contexts and application domains.

As future work, we plan to extend the metric set to cover additional user characteristics. We will also involve user-centered design experts to assess the usability and effectiveness of the proposed abstraction and adaptation process. Furthermore, we plan to abstract the remaining user characteristics and automate the measurement process by leveraging models as executable artefacts,

analysing whether existing models are sufficiently expressive or require additional annotations to support metric extraction. Finally, we intend to validate the approach with end users through empirical studies, including usability testing, to evaluate its understandability and practical applicability.

Acknowledgments. This work was supported by the Generalitat Valenciana with TENTACLE under project CIAICO/2023/089, and with the Spanish Ministry of Science under the projects PHYLOVAR (PID2024-162114OB-I00) and EU4IRI (PID2024-161104OB-C22), financed by MICIU/AEI/10.13039/501100011033 and by FEDER.

References

1. Abb, L., Rehse, J.: Process-related user interaction logs: State of the art, reference model, and object-centric implementation. Inf. Syst. **124**, 102386 (2024). https://doi.org/https://doi.org/10.1016/j.is.2024.102386
2. Barra, S., Carta, S.M., Giuliani, A., Pisu, A., Podda, A.S., Riboni, D.: Footapp: An ai-powered system for football match annotation. Multim. Tools Appl. **82**(4), 5547–5567 (2023). https://doi.org/10.1007/s11042-022-13359-0
3. Ben, W., Jiqun, L.: Characterizing and early predicting user performance for adaptive search path recommendation. In: Proceedings of the Association for Information Science and Technology. vol. 60, pp. 408–420 (2023). https://doi.org/10.1002/pra2.799
4. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool Publishers, California, USA (2017). https://doi.org/10.2200/S00751ED2V01Y201701SWE004
5. Brdnik, S., Hericko, T., Sumak, B.: Intelligent user interfaces and their evaluation: A systematic mapping study. Sensors **22**(15), 5830 (2022). https://doi.org/10.3390/s22155830
6. d'Aloisio, G., Sipio, C.D., Marco, A.D., Ruscio, D.D.: How fair are we? from conceptualization to automated assessment of fairness definitions. CoRR (2024). https://doi.org/10.48550/arXiv.2404.09919
7. Darejeh, A., Marcus, N., Mohammadi, G., Sweller, J.: A critical analysis of cognitive load measurement methods for evaluating the usability of different types of interfaces: guidelines and framework for human-computer interaction. CoRR (2024). https://doi.org/10.48550/arXiv.2402.11820
8. Gaspar, A., Panach, J.I., Gil, M., Romero, V.: Propuesta de métricas de usuario para definir perfiles de usuario. Revista de la Asociación Interacción Persona Ordenador (AIPO) **6**(1), 37–50 (2025)
9. Goundar, M.S., Kumar, B.A., Ali, A.B.M.S.: Development of usability guidelines: A systematic literature review. Int. J. Hum. Comput. Interact. **40**(5), 1298–1316 (2024). https://doi.org/10.1080/10447318.2022.2141009
10. Guo, J., Liu, Z., Tian, S., Huang, F., Li, J., Li, X., Igorevich, K., Ma, J.: Tfl-dt: A trust evaluation scheme for federated learning in digital twin for mobile networks. IEEE Journal on Selected Areas in Communications **41**(11), 3548–3560 (2023). https://doi.org/10.1109/JSAC.2023.3310094
11. IntegraNova: (2025), https://www.integranova.com, accessed December 18, 2025
12. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V.: Usixml: A language supporting multi-path development of user interfaces. In: Engineering Human Computer Interaction and Interactive Systems: Joint Working Conferences EHCI-DSVIS 2004. pp. 200–220. Springer (2005)

13. Meixner, G., Schnitzler, A.P., Seissler, M., Reichart, M., Härtl, J., Creßer, A.: Model-based user interface development for smart homes: Different viewpoints on a common theme. In: Proceedings of the International Conference on Human-Computer Interaction (HCI). vol. 6761, pp. 177–186. Springer, Orlando, USA (2011). https://doi.org/10.1007/978-3-642-21602-2_22
14. Moured, O., Baumgarten-Egemole, M., Müller, K., Roitberg, A., Schwarz, T., Stiefelhagen, R.: Chart4blind: An intelligent interface for chart accessibility conversion. In: Proceedings of the 29th International Conference on Intelligent User Interfaces, IUI 2024, Greenville, SC, USA, March 18-21, 2024. pp. 504–514. ACM (2024). https://doi.org/10.1145/3640543.3645175
15. Palma, S.D., Nucci, D.D., Palomba, F., Tamburri, D.A.: Toward a catalog of software quality metrics for infrastructure code. J. Syst. Softw. **170**, 110726 (2020). https://doi.org/10.1016/j.jss.2020.110726
16. Paternò, F., Santoro, C., Spano, L.D.: Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Transactions on Computer-Human Interaction (TOCHI) **16**(4), 1–30 (2009). https://doi.org/10.1145/1614390.1614394
17. Quico, F.J.V., Battisti, A.L.É., Muchaluat-Saade, D.C., Delicato, F.C.: L-PRISM: A domain-specific language for describing multimedia service function chains. J. Braz. Comput. Soc. **31**(1), 545–569 (2025). https://doi.org/10.5753/jbcs.2025.5453
18. Rao, N., Bansal, C., Mukherjee, S., Maddila, C.S.: Product insights: Analyzing product intents in web search. In: CIKM '20:. pp. 2189–2192. ACM, Ireland (2020). https://doi.org/10.1145/3340531.3412090
19. Rasch, J., Middelbeck, D.: Knowledge state networks for effective skill assessment in atomic learning. CoRR (2021). https://doi.org/10.48550/arXiv.2105.07733
20. Sharma, B., Pokharel, P., Joshi, B.: User behavior analytics for anomaly detection using lstm autoencoder - insider threat detection. In: Proceedings of the 11th International Conference on Advances in Information Technology (IAIT 2020). pp. 1–9 (2020). https://doi.org/10.1145/3406601.3406610
21. Shen, Q., Wen, H., Tao, W., Zhang, J., Lv, F., Chen, Z., Li, Z.: Deep interest highlight network for click-through rate prediction in trigger-induced recommendation. In: Proceedings of the ACM Web Conference, Lyon, France, April 25 - 29, 2022. pp. 422–430 (2022). https://doi.org/10.1145/3485447.3511970
22. Stephanidis, C., Antona, M., Gao, Q., Zhou, J.: HCI International 2020–Late Breaking Papers: Universal Access and Inclusive Design: 22nd HCI International Conference, HCII 2020., vol. 12426. Springer Nature, Copenhagen, Denmark (2020)
23. Wiedner, M., Fatol, A., Furrer, A., Eisemann, L., Frazzoli, E.: CARSI 3.0: A context-driven intelligent user interface. In: AutomotiveUI 2025, Brisbane, Australia. pp. 287–293. ACM (2025). https://doi.org/10.1145/3744335.3758519
24. Williford, B., Runyon, M., Hammond, T.: Recognizing perspective accuracy: an intelligent user interface for assisting novices. In: Proceedings of the 25th International Conference on Intelligent User Interfaces. pp. 231–242 (2020)
25. World Wide Web Consortium (W3C): Abstract User Interface (AUI). https://www.w3.org/TR/abstract-ui/, accessed: 2024-09-06
26. Yu, R., Tang, R., Rokicki, M., Gadiraju, U., Dietze, S.: Topic-independent modeling of user knowledge in informational search sessions. Information Retrieval Journal **24**(3), 240 – 268 (2021). https://doi.org/10.1007/s10791-021-09391-7
27. Zhou, J., Zahiri, S.M., Hughes, S., Jadda, K.A., Kallumadi, S., Agichtein, E.: Debiased modeling of search click behavior with reinforcement learning. In: The 44th International Conference on Research and Development in Information Retrieval. pp. 1637–1641. ACM, Canada (2021). https://doi.org/10.1145/3404835.3463228