Enhancing Flexibility in User Interaction Modeling by Adding Design Uncertainty to IFML

Marco Brambilla* Romina Eramo† Alfonso Pierantonio† Gianni Rosa† Eric Umuhoza†

* Department of Electronics, Information and Bioengineering. Politecnico di Milano, Italy
Email:{name.surname}@polimi.it

† Department of Information Engineering, Computer Science and Mathematics. Università degli Studi dell'Aquila, Italy
Email:{name.surname}@univaq.it

***Abstract*—User interaction (UI) design is an important task in the development of software applications: in fact the success of the application itself, as well as the business behind it, is strongly related to the user experience. Unfortunately, designers can obtain realistic feedback from users about their actual expectations only at runtime, by analyzing the user behavior over the final application.**

A possible solution to this problem is to integrate the user feedback in the design phase, for example through A/B split testing, which allows to test the effectiveness of different variants of the application interface. However, so far A/B testing has been addressed only through manual coding and a-posteriori refactoring based on the analysis of the results. Model-driven development may enable the integration of such techniques with runtime log analysis and design-time application specifications. Unfortunately, creating new alternatives in the model usually corresponds to a combinatorial explosion of the application versions, making the testing hard to manage.

In this paper, we propose a model-driven approach that enables to denote design alternatives in a compact way by adopting a model for uncertainty, integrated with a model for the user interaction design. Thus, the multiple possibilities can be represented by a single user interaction model (i.e., IFML model) from which a single software application will be generated, including all the variations that need to be evaluated. Uncertainty can then be solved by integrating the results of user behavior analysis (for instance, over the application logs of a web site). Thanks to this, our approach considerably reduces the costs of the user interaction optimization.

I. Introduction

Over the last decade and half, modeling techniques have been leveraged to cover requirement specification [3], [2], design [28], and verification/validation (e.g., [18]) of software artifacts. In the context of user interaction (UI) modeling, model-driven techniques have been proposed (see [6], [5]) for a wide range of tasks, including model checking, full code generation of the designed application, runtime user tracking and integration with design-time models.

One of the core aspects of UI design with respect to other system perspectives, is that designers must aim at guaranteeing a better user experience (e.g., the user expects to find the desired information in the expected place and through intuitive navigation paths), as this is key to the acceptance and adoption of the designed application, as well as to the success of the underlying business.

The Interaction Flow Modeling Language (IFML) [8], [7], [10] is designed for expressing the content, user interaction and control behavior of software applications. Unfortunately, 'the proof of the pudding is in the eating': designers are given relevant feedback about the actual expectations only by analyzing the user behavior at runtime. Hence, some a-posteriori adaptations of the UI could be needed for meeting the application goals. A way to address such difficulties consists in reducing the gap between UI design and user experience by involving users in the design phase, for example through A/B split testing [24], in which different end-users examine different alternatives, and the most effective ones are selected based on statistical analysis of user behavior.

This can be viewed as a form of bound design uncertainty [14]: while a number of alternative solutions have been contemplated, the designer does not hold enough information or knowledge to decide at this stage of the process which is the right one. A trivial strategy would consist in creating a different IFML model for each alternative to be evaluated. However, this implies various disadvantages: first, a small set of alternatives in the model usually corresponds to a combinatorial explosion of the application versions, making the testing hard to manage due to the difficulty to precisely back propagate user feedback to the right model. Second, by designing independent models the designer misses all the logics that covers the decisions on which alternatives to show to which user, which runtime data to collect, and what kind of analysis to run on the user behaviour to decide which variant should be preferred.

In this paper, we propose an extension to the IFML modeling notation, which leverages uncertainty to a first-class status. Uncertainty-related decisions are documented in an intentional manner by means of models with uncertainty, which permit the designer to make transitional decisions to be either confirmed or undone in a traceable way once enough data analytics about user behavior are available.

The paper is organized as follows: Sect. II discusses the background on the IFML language. In Sect. III, an example clarifies the addressed problem. The extension to uncertainty for IFML is presented in Sect. IV. Finally, related work is discussed in Sect. V and conclusions are drawn in Sect. VI.

II. Background: IFML

The Interaction Flow Modeling Language (IFML) is designed for expressing the content, user interaction and control behavior of the front-end of software applications. Its metamodel uses the basic data types from the UML metamodel, specializes a number of UML metaclasses as the basis for IFML metaclasses, and presumes that the IFML Domain Model is represented in UML.

An IFML model supports the following design perspectives: The *view structure specification*, which consists of the definition of view containers, their nesting relationships, their visibility, and their reachability; The *view content specification*, which consists of the definition of `ViewComponents`, i.e., content and data entry elements contained within `ViewContainers`; The *events specification*, which consists of the definition of `Events` that may affect the state of the user interface. `Events` can be produced by the user's interaction, by the application, or by an external system; The *event transition specification*, which consists of the definition of the effect of an `Event` on the user interface; The *parameter binding specification*, which consists of the definition of the input-output dependencies between `ViewComponents` and between `ViewComponents` and `Actions`; and The reference to `Actions` triggered by the user's events. The effect of an `Event` is represented by an `InteractionFlow` connection, which connects the event to the `ViewContainer` or `ViewComponent` affected by the `Event`. The `InteractionFlow` expresses a change of state of the user interface: the occurrence of the event causes a transition of state that produces a change in the user interface.

IFML concepts can be stereotyped to describe more precise behaviors. For instance, one could define specific stereotypes for describing web pages (a specific kind of `ViewContainer`); forms, lists and details (specific kinds of `ViewComponent`); and submission or selection `events`. By exploiting this extensibility feature, we defined custom extensions of IFML for covering Web [1] and mobile [9], [25] applications.

Figure 1*(a)* shows a piece of a user interface that allows to search an album by title or year, visualize a list retrieved albums, and see details of a selected album. Figure 1*(b)* shows the corresponding IFML model which consists of: (i) a `ViewContainer` *AlbumSearch* which contains a specific `ViewComponent`, a *Form* named Album Search. The form is associated with a submission `Event`; (ii) a `ViewContainer` *Albums* which contains a `ViewComponent`, *List* named Album List; and (iii) a `ViewContainer`, Album containing a `ViewComponent`, *Details* named Album Details.

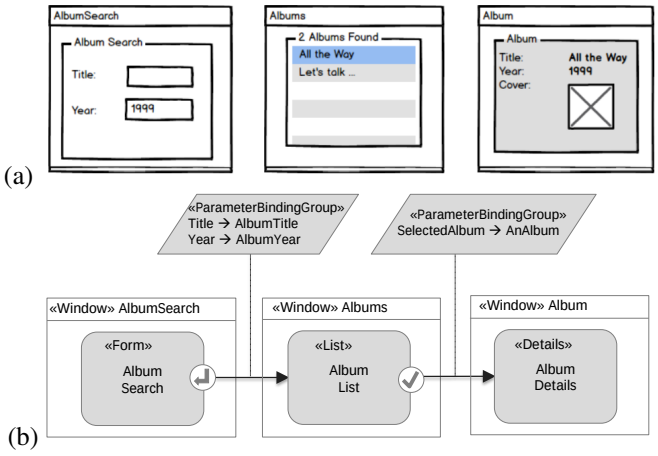

Fig. 1: IFML Example: *(a)* a piece of user interface and *(b)* the corresponding IFML model. The user enters data into an input *form* and submits; the submission *event* causes a distinct *ViewContainer* to appear with a *list* of matching albums; the selection of an item in the list causes the display of the corresponding *details* in a third *ViewContainer*.

III. Problem Statement

An interaction model is a design model that describes the front-end of a software application with the aim to bind the contents together in a way that supports the conceptual models of its target users. Thus, interaction design decisions have to meet customer preferences. As said, since testing and understanding the user behavior will be possible when the application is running, the designer could be uncertain about the better strategy to adopt in the design of user interaction. This can be translated in a set of design variants describing slightly different user interactions. Such design alternatives have to be compared and evaluated in terms of user acceptance, usability, and effectiveness from a business perspective (e.g., leading to more or less transactions performed by a user).

Given such issues, we aim at finding the most compact and efficient way for describing these variants, so as to:

- maximize traceability of all the phases of the development;
- minimize the risk of inconsistency across the variants;
- integrate the generation of the code for the management of the different variants and the selection of the appropriate variant to be shown to the different users;
- simplify the integration of the modeling of the variant with the runtime data collection describing the user behaviour across the variants;
- enable integration of statistical analysis of runtime data and final selection of the best variants with no additional cost of development.

In order to make the problem statement more concrete, we now describe a simple running example which will be used along the whole discussion.

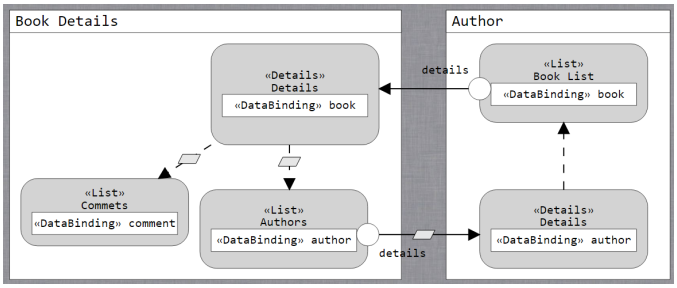

Fig. 2: **Version 1 of Book Details**. The details of a selected book are displayed along with two lits: comments and authors of the selected book.

A. Example: Bookstore

Bookstore is a kind of e-commerce web application for book selling. The books are organized in categories. A book is associated with its authors, comments, and editorial reviews. The main functionalities of the applications include: (i) Book searching which allows the user to search for a book, and visualize its details (including the book's authors and comments related to the selected book); (ii) Book review which allows the user to write a comment on a book; and (iii) purchasing, which allows to buy a desired book.

Figure 2 shows a piece of the user interaction model allowing the user to visualize details of a selected book, in *Book Details* Page. The details of the Book are displayed along with two lists: a list of comments and another one for the authors of the selected book. The user can access *Author* page to see detailedc information (including other books he co-authored) of the selected author. In Figure 3, we show an alternative way to display the same content as in Figure 3. The UI in Figure 3 is obtained from the one in Figure 2 through the following actions:

- removal of a `List` of *authors* from the `Page` *Book Details*
- addition of a new `Page`, *Authors*, containing the list of authors of a selected book
- addition of a new `event`, called *authors*, allowing the user to visualize the `List` of authors of the selected book

To decide which version of the UI to keep for the final implementation, one can conduct the experiments in which end-users are directly asked to provide their preferences on these two alternatives. However, little new alternatives in the model usually correspond to a combinatorial explosion of the application versions, making the testing hard to manage.

IV. Representing Uncertainty in IFML

Without dedicated support, working with uncertainty means that modelers handle each alternative model separately. In order to reduce the burden of managing a collection of models, we consider uncertainty as a first-class concern in the design, implementation, and deployment of those systems [17]. To this end, the metamodel-independent approach discussed in [13] able to manage a set of variants by means of a model with uncertainty is exploited. The metamodel generation is realized

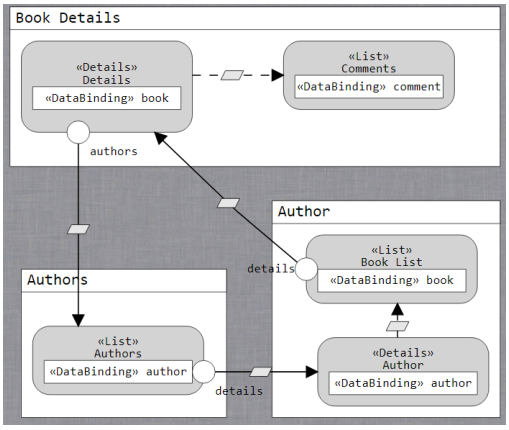

Fig. 3: **Version 2 of Book Details**. The details of a selected book are displayed along with a list of its comments. The list of authors is placed in a new page, *Authors*, reachable from Book Details page via a new event, *authors*.

by means of a model-to-model transformation `M2UM` written in ATL[1]. It takes as input a metamodel `M` conforming to Ecore[2] and generates the corresponding metamodel with uncertainty `UM`. At this point, modelers may specify their models with uncertainty conform to `UM`. An `um` model is semantically equivalent to a set of models $m_1, ..., m_n$ conforms to `M` (i.e., all the candidate solution models represented in `um`).
The next sections illustrate in detail the uncertainty metamodel and how it can be generated and utilized.

A. Uncertainty IFML Metamodel

The uncertainty metamodel UM is obtained by extending a *base* metamodel M with specific connectives able to represent the multiple outcomes of a transformation. These connectives denote the uncertainty points where alternative model elements are attached. Let us consider the IFML language presented in Sec. II. [3]. For the sake of brevity, we provide a restricted version of the IFML metamodel that allows us to model the case study discussed in Sect. III.

The uncertainty metamodel *UIFML* is depicted in Fig. 4 as an extension of the considered restricted *IFML*. In particular, the uncertainty metamodel UIFML is generated extending the base metamodel as follows:

1) the abstract metaclass `TracedClass` with references `include` and `exclude` is added;
2) the metaclass `OperatorType` that enumerates the logical operator literals `AND`, `OR` and `XOR` is added;
3) the abstract metaclass `UClass`, enabling the representation of uncertain elements, is added. It has the attributes `name`; the attribute `uType` of type `OperatorType` for specifying the logical operator connecting alternatives;
4) for each metaclass *C* in base *IFML*, such that it does not specialize other metaclasses:

[1]ATL Transformation Language: https://eclipse.org/atl/
[2]EMF Ecore: http://www.eclipse.org/modeling/emf/
[3]For reason of readability, we omit the description of the language. The interested reader may refer to the OMG specification

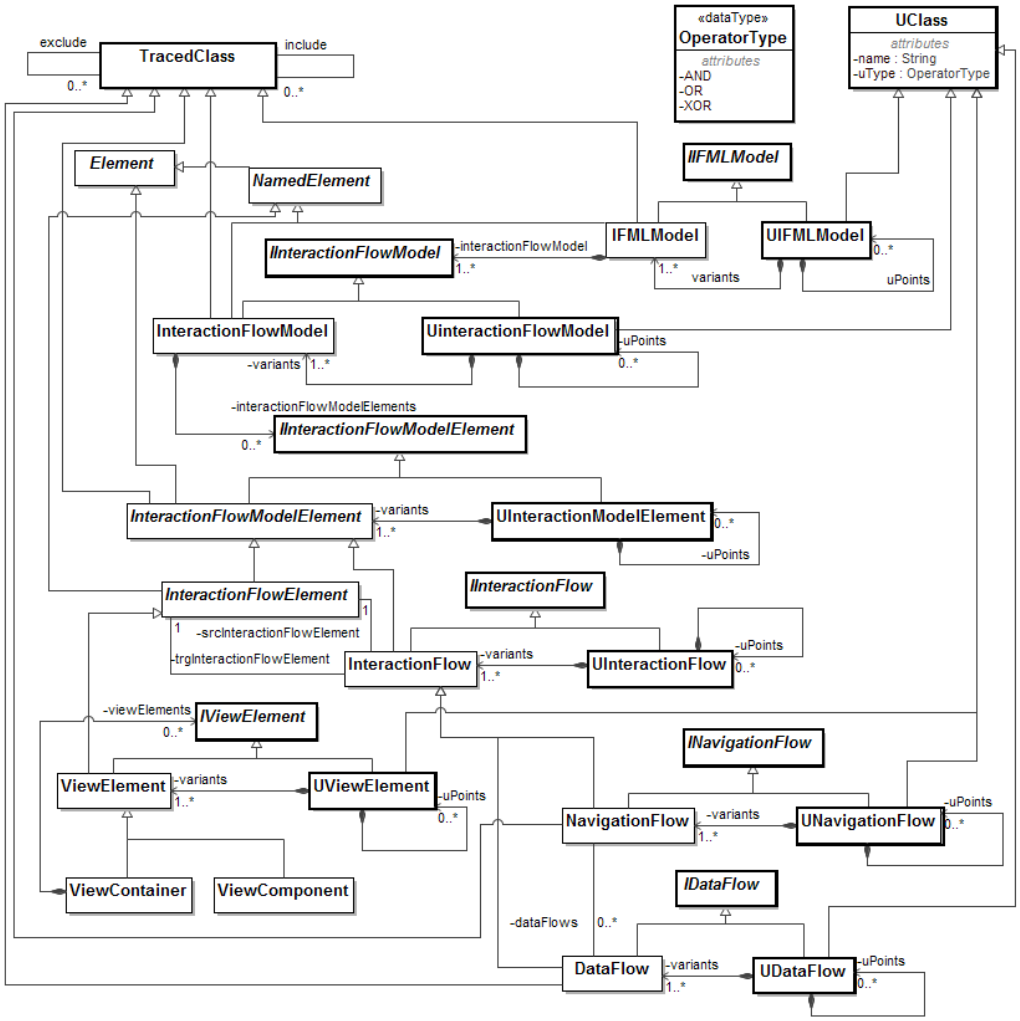

Fig. 4: IFML Uncertainty metamodel

4.1) a corresponding abstract metaclass `IC` is created such that `C` specializes `IC`. Such interface allows to choose among a certain element or an uncertain one

4.2) a metaclass `UC` that extends `IC` and `UClass` is added. It specifies an association `variants` of type `C` and multiplicity `1..*` that relates the alternative elements to the uncertainty point of type `C` and an association `uPoints` of type `UC` and multiplicity `0..*` enabling to nest uncertainty points.

4.3) the metaclass *C* is generalized by `TracedClass`,

The main advantage of such representation technique is that the uncertainty represented by a set of alternative models is leveraged to a first-class status. Hence, a complete set of models can therefore be manipulated as whole, for instance with an automated transformation (as done for instance in [16]), without iterating over each individual in the set.

B. IFML Model with Uncertainty

At this point, modelers may specify their user interaction models with uncertainty. In particular, by using the aforementioned approach, the model in Fig. 2 and Fig. 3 can be represented by means of a unique model with uncertainty depicted in Fig. 5. The model is composed of a common part and an uncertain part denoted by dashed lines.

In particular, according to the example in Sect. III we have two Uncertainty Points called respectively **V1** and **V2**. Each uncertainty point is composed of a set of elements that are in common with the chosen version. For instance, the uncertainty point **V1** includes: a `ViewComponent`, List of Authors; an `Event` allowing to see the details of the selected author; and a `Navigation Flow` from Book Details Page to the Author Page (containing the details of a selected author). While the uncertainty point **V2** includes: `Event` named authors, which allows to access the list of authors of the selected book; `ViewContainer`, a page named Authors. It contains a `ViewComponent` named Authors, displaying the list of authors of the selected book; a `Navigation Flow` from Book Details Page to Authors Page; An `Event` named details, leading to detailed information of the selected author; and a

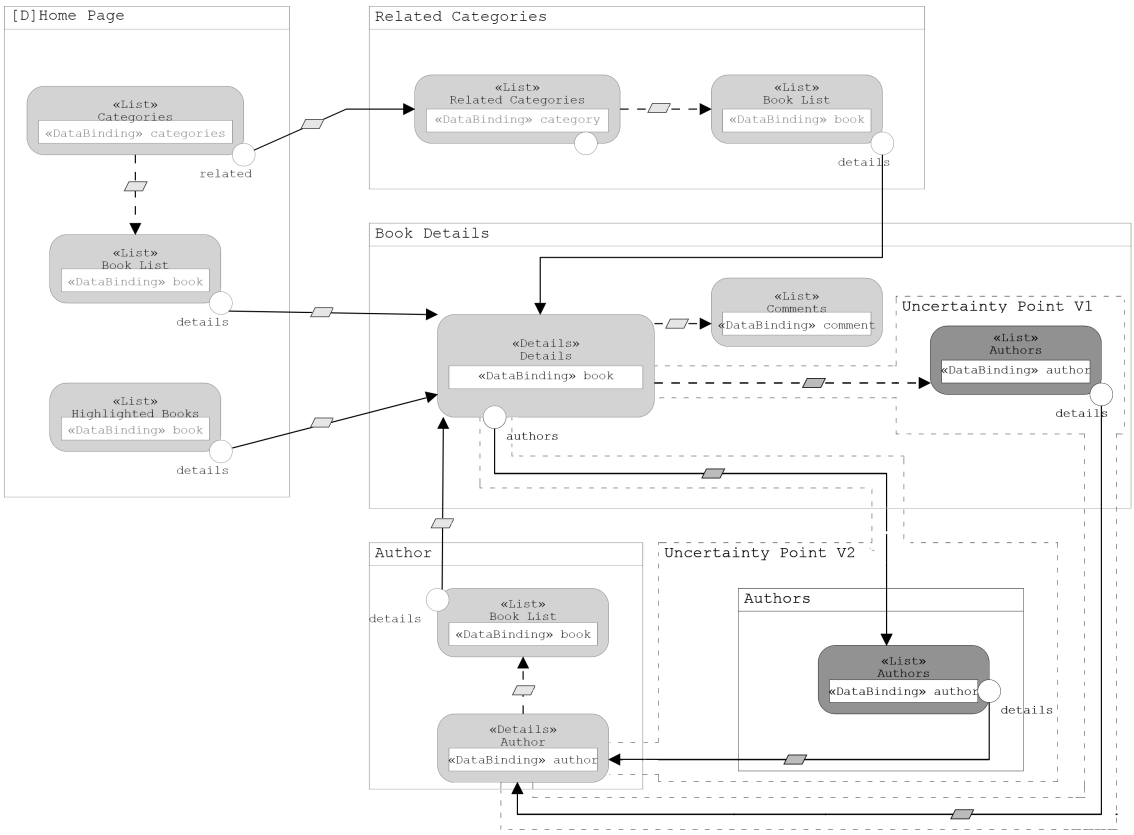

Fig. 5: IFML Uncertainty Model

`Navigation Flow` from the page of authors to the page named Author, which contains the details selected author.

V. Related Work

a) Conceptual Modeling of WebApplications: This work is widely related to a large corpus of researches that address conceptual modeling of Web applications. Among them we can cite: (i) Araneus [20], a modeling proposal for Web applications that allows one to represent the hypertext organization, with nested relations and inter-page links; (ii) W2000, formerly HDM [4], which introduced a notion of model-based design, clearly separating the activities of authoring in-the-large (hypertext schema design) and authoring in-the-small (page and content production); (iii) OO-HDM [23], a UML-based approach for modeling and implementing Web application interfaces; (iv) Hera [26], a model-driven design approach and specification framework focusing on the development of context-dependent or personalized Web information system; (v) Web Application Extension for UML (WAE) [11], a UML extension for describing Web application interfaces and the client-server interactions; (vi) WebDSL [27], a domain-specific language consisting of a core language with constructs to define entities, pages and business logic, plus extensions.

b) Design Uncertainty: Uncertainty is ubiquitous within contexts such as requirements engineering [12], software processes [19] and adaptive systems [22]. Uncertainty management has been studied in many works, often with the intention to express and represent it in models. In [15], the notion of *partial model* is introduced in order to let the designer specify uncertain information by means of a base model enriched with annotations and first-order logic. Model transformation techniques typically operate under the assumption that models do not contain uncertainty. Nevertheless, the work in [16] proposes a technique for adapting existing model transformations in order to deal with models containing uncertainty. In particular, a lifting operation permits to adapt unidirectional transformations for being used over models with uncertainty preserving their original behavior. In [21] a formal approach called MAVO is proposed and applied to design models in order to express and allow automated reasoning in presence of uncertainty.

VI. Conclusion and Future Work

In this paper we proposed a model-driven approach that enables to denote design alternatives in a compact way by adopting a model for uncertainty, integrated with a model for the user interaction design. In this way, the multiple possibilities are represented in a single user interaction model from which a single software application can be generated, including all the variations that need to be evaluated.

As future work, we plan to provide a visual editor for the specification of models with uncertainty with graphical

and concrete syntaxes depending on the tool availability for the considered modeling language. In our case, the adopted syntaxes for specifying models do not affect the overall approach since models are manipulated by considering their abstract syntaxes. Whereas, a specialized visualization of the uncertainty will allow the modeler to benefit from hiding, partitioning, constraining, and abstracting while traversing an uncertain model. Furthermore, we plan to manage the uncertainty by integrating the proposed approach with other approaches for user behavior analysis like the one proposed in [6].

REFERENCES

[1] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti. Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform. In *Proceedings of ICWE 2015*, pages 605–608, 2015.

[2] Ameller and E. Al. Handling non-functional requirements in Model-Driven Development - An ongoing industrial survey. *RE*, pages 208–213, 2015.

[3] D. Amyot and G. Mussbacher. User requirements notation: The first ten years, the next ten years. *Journal of Software*, 6(5):747–768, July 2011.

[4] L. Baresi, F. Garzotto, P. Paolini, and P. Paolini. From web sites to web applications: New issues for conceptual modeling. In *ER (Workshops)*, pages 89–100, 2000.

[5] C. Bernaschina, M. Brambilla, T. Koka, A. Mauri, and E. Umuhoza. Integrating modeling languages and web logs for enhanced user behavior analytics. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 171–175, 2017.

[6] C. Bernaschina, M. Brambilla, A. Mauri, and E. Umuhoza. A big data analysis framework for model-based web user behavior analytics. In *International Conference on Web Engineering*, pages 98–114, 2017.

[7] M. Brambilla and P. Fraternali. *Interaction Flow Modeling Language: Model-driven UI engineering of web and mobile apps with IFML*. Morgan Kaufmann, 2014.

[8] M. Brambilla, P. Fraternali, et al. The interaction flow modeling language (IFML). Technical report, version 1.0., The Object Management Group (OMG), http://www.ifml.org.

[9] M. Brambilla, A. Mauri, and E. Umuhoza. Extending the interaction flow modeling language (ifml) for model driven development of mobile applications front end. In *International Conference on Mobile Web and Information Systems*, pages 176–191. Springer, 2014.

[10] M. Brambilla, E. Umuhoza, and R. Acerbis. Model-driven development of user interfaces for iot systems via domain-specific components and patterns. *Journal of Internet Services and Applications*, 8(1):14, 2017.

[20] G. Mecca, P. Merialdo, P. Atzeni, V. Crescenzi, and V. Crescenzi. The (short) araneus guide to web-site development. In *WebDB*, pages 13–18, 1999.

[11] J. Conallen. *Building Web applications with UML*. Addison Wesley, 2002.

[12] C. Ebert and J. D. Man. Requirements uncertainty: influencing factors and concrete improvements. In *Procs. of ICSE*, pages 553–560. ACM Press, 2005.

[13] R. Eramo, A. Pierantonio, and G. Rosa. Managing uncertainty in bidirectional model transformations. In *SLE 2015*, pages 49–58, 2015.

[14] M. Famelis and M. Chechik. Managing design-time uncertainty. *Software & Systems Modeling*, 54(9):69, Mar. 2017.

[15] M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *ICSE*, pages 573–583, 2012.

[16] M. Famelis, R. Salay, A. D. Sandro, and M. Chechik. Transformation of models containing uncertainty. In *MoDELS'13*, pages 673–689, 2013.

[17] D. Garlan. Software engineering in an uncertain world. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 125–128. ACM, 2010.

[18] H. Giese, M. Tichy, S. Burmester, and W. Schäfer. Towards the compositional verification of real-time UML designs. *ACM SIGSOFT*, 28(5):38, 2003.

[19] H. Ibrahim, B. H. Far, A. Eberlein, and Y. Daradkeh. Uncertainty management in software engineering: Past, present, and future. In *CCECE*, pages 7–12. IEEE, 2009.

[21] R. Salay, M. Chechik, J. Horkoff, and A. D. Sandro. Managing requirements uncertainty with partial models. *Requir. Eng.*, 18(2):107–128, 2013.

[22] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. Requirements-aware systems: A research agenda for re for self-adaptive systems.

[23] D. Schwabe, G. Rossi, and G. Rossi. The object-oriented hypermedia design model. pages 45–46, 1995.

[24] M. Speicher, A. Both, and M. Gaedke. Ensuring Web Interface Quality through Usability-Based Split Testing. In *ICWE 2014*.

[25] E. Umuhoza, H. Ed-douibi, M. Brambilla, J. Cabot, and A. Bongio. Automatic code generation for cross-platform, multi-device mobile apps: Some reflections from an industrial experience. In *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*, MobileDeLi 2015, pages 37–44, New York, NY, USA, 2015. ACM.

[26] R. Vdovják, F. Frăsincar, G.-J. Houben, and P. Barna. Engineering Semantic Web Information Systems in Hera. *Journal of Web Engineering*, 1(1-2):3–26, 2003.

[27] E. Visser. WebDSL: A case study in domain-specific language engineering. In *GTTSE 2007*, 2008.

[28] M. Voelter and E. al. *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013.