Improving UML Support for User Interface Design: A Metric Assessment of UML*i*

Paulo Pinheiro da Silva[1] and Norman W. Paton[2]

[1]Department of Computer Science, Stanford University
Gates Building, Stanford, CA 94305, USA.
E-mail: pp@ksl.stanford.edu
[2]Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, England, UK.
E-mail: norm@cs.man.ac.uk

October 23, 2002

Abstract

The Unified Modeling Language (UML) has been widely accepted by application developers, but not so much by user interface (UI) designers. For this reason, the Unified Modeling Language for Interactive Systems (UML*i*) has been proposed to improve UML support for UI design. UML*i* introduces a diagram notation for modeling UI presentation and extends activity diagram notation to describe collaboration between interaction objects and domain objects. This paper demonstrates using design metrics, in a quantitative way, that UML*i* models are significantly structurally, behaviorally and visually less complex than standard UML models when describing the same set of properties of an interactive system.

1 INTRODUCTION

The development of techniques for constructing UI models and generating running UIs from these models has been studied since user interface management systems (UIMSs) were proposed during the 80s. More recently, model-based user interface development environments (MB-UIDEs) [8, 23] have provided additional experience on the construction of user interface models. The use of many different and sometimes user interface specific notations is one of the well-known shortcoming of MB-UIDEs [19]. Indeed, the modest adoption of MB-UIDEs as part of software development practice can at least partly be ascribed to the difficulties associated with the integration of MB-UIDEs with mainstream application development techniques. Therefore, the idea of modeling UIs using UML [16], a standard modeling notation in widespread use for modeling mainstream systems, has attracted the attention of both HCI and software engineering communities [13, 15, 18, 20].

UML, however, suffers from its lack of support for modeling UIs [11, 18]. For example, class diagrams are not entirely suitable for modeling UI presentation. As a result of these difficulties, research has taken place with a view to improving the effectiveness of UML for UIs. For example, Markopoulos's approach [13], Wisdom [15] and UML*i* [20] are conservative extensions of UML, while ConcurTaskTree [18] that proposes the introduction of a new notation for task modeling constructors into standard UML. By contrast, the notion of task is represented by classes in Wisdom and by activities in UML*i* and Markopoulo's approach. The existence of more than one approach to improving UI support in UML indicates the necessity of evaluating the benefits of such approaches compared with standard UML. Indeed, it is still unclear how best to improve support for UI design in the context of UML.

Qualitative evaluations based on the verification of the conformance of proposed improvements with design practices and guidelines are supported by most of these approaches. For instance, [13], [18] and [20] are approaches derived from MB-UIDEs. However, such qualitative evaluations do not identify the extent of the benefits achieved. Therefore, this paper presents, for the first time, a quantitative assessment of the benefits of extensions to UML for modeling interactive systems. This assessment is described in terms of the structural and behavioral complexity of UML*i* and UML models for interactive systems.

2 RELATED WORK

Few assessment strategies for UI designs evaluate the quality of conceptual UI models. Considering toolkits and UI builders, no conceptual specification of UIs is usually produced at all. Indeed, the produced UI codes are already concrete UI presentations, generally committed to a specific layout and a selected set of widgets. Considering UIMSs and MB-UIDEs, an interaction with generated UI prototypes and UI codes allows designers and users to evaluate the quality of a combination of UI models, techniques used to generate running UIs, and toolkits that may be used to implement the running UIs rather than the quality of the UI models in isolation. Some quality evaluation can be achieved by the simulation of these conceptual models, as in ConcurTaskTree [17]. Simulations, however, restrict assessment to the behavioral part of the models.

The quality of models, however, can be assessed from internal attributes of the models rather than from attributes of artifacts produced from them. For instance, object-oriented design metrics [1, 3, 5, 6, 14] have been used to evaluate the quality of several aspects, such as the collaboration (coupling) between classes [5, 6], the cohesion among the operations of a class [3, 5], and the complexity of the control-flow [14] of as object-oriented design.

Therefore, object-oriented design metrics are used in this paper to assess the benefits of using UML*i* to model an interactive system when compared with the use of standard UML. In terms of structural complexity, this study uses the suite of metrics proposed by Chidamber and Kemere (CK metrics) [6] because:

- Basili et. al. [1] have validated CK metrics for class fault-proneness [1] (disregarding the *lack of cohesion on methods* LCOM), and Li and Henry [12] have validated CK metrics against the number of maintenance modifications.

- Basili et. al. [1] have provided a valuable indication of the impact of UI classes (or *widgets*) on the the validation of each CK metrics. Indeed, the experimentation described there was composed of eight medium-sized interactive systems built in C++. Further, the user interfaces of those systems were built using the OSF/MOTIF toolkit. Therefore, due to the controlled nature of the experiment, it was possible to describe the impact of CK metrics on the database and user interface classes, the two categories of classes described in the paper. Furthermore, the comments in [1] are contrasted to the CK metrics of the case study in this paper.

- It has influenced many other metric proposals [3].

In terms of behavioral complexity, this study considers the McCabe's cyclomatic complexity [14] since it is a long-term well-established metric the definition of which has been naturally translated from a code-level metric to a design-level metric.

3 CASE STUDY

The modeling of a library system in both UML and UML*i* is used as the case study in this paper. This section characterizes the aspects of a generic interactive system that are commonly modeled in UIMSs and MB-UIDEs. From this characterization, a brief description of the library system modeled in the following sections is provided.

3.1 Aspects of Interactive Systems

Many aspects of an interactive systems can be described by models. Therefore, there are many possible combinations of models that can together describe an interactive system. Despite how different these models can be, structural and behavioral aspects of systems should be considered if the aim is to build UI models that can be used, for instance, to generate running user interfaces.

- *Structural models.* Classes and objects are the main structural elements of a system. Relationships between classes and objects, i.e., associations, compositions and generalizations, are also structural elements. Thus, structural models describe properties of classes, objects and their relationships.

  - *Presentation models*. Classes and objects responsible for the visual appearance of user interfaces are structural elements. Interaction objects are usually called *widgets*. Presentation models are structural models describing
properties of widgets and their classes.

  - *Domain models.* Classes and objects modeling the entities of a system are elements of the domain. Many of them may be related to interaction classes and widgets, despite the fact that they are not inherently related to

UIs. Thus, domain models describe properties of classes and objects of the domain.

- *Behavior models*. Dynamic elements used to alter the states of structural elements, i.e., tasks, actions, events, are behavioral elements of a generic system. Thus, behavioral models describe properties of behavioral elements.

A description of how domain, presentation and behavioral models for the library system can be built using UML and UML*i* are discussed in the following sections. First, however, the functionalities considered in the case study are described.

3.2 The Library System Case Study

`Books`, `BookCopies` and `LibraryUsers` classified into `Borrowers` and `Librarians` are the main entities of the library system. Considering these entities, the use case diagram in Figure 1 identifies the top-level functionalities specified for the library system. There, `LibraryUsers` are entitled to connect to the system (`Connect` use case), search for books (`SearchBook` use case) and check the status of a book (`CheckBookStatus` use case). `Librarians` can additionally check books in (deleting loans), check books out (creating loans), renew loans, and maintain the `Book`, `BookCopy` and `LibraryUser` catalogues.

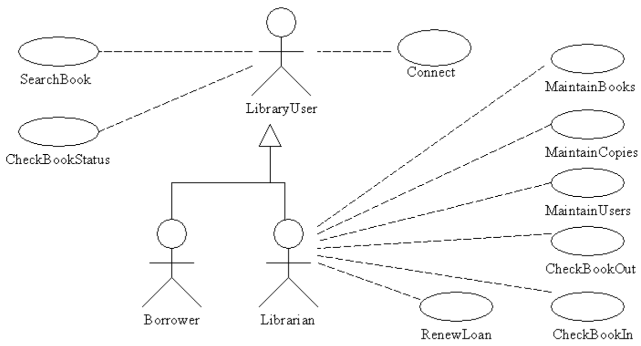

Figure 1: The use case diagram of the Library System.

Many other functionalities are not considered in the case study. For instance, users are not allowed to browse the book catalogue, to get a list of their loans or to make a book reservation. However, this simple specification has provided sufficient functionality to require the construction of quite substantial models for use in the study using metrics. Indeed, the library system must be entirely modeled in order to make the quantitative assessment a result of *the effort of building complete models of interactive systems*.

UML and UML*i* models of the library system are partially described in the following two sections. Documentation containing a complete description of these models along with the actual UML and UML*i* models is available at `http://img.cs.man.ac.uk/umli/metrics`. Moreover, the models can be viewed and adapted using Argo/UML*i* [21], which is a UML editor tool that implements the UML*i* extensions and is also publicly available from `http://img.cs.man.ac.uk/umli/software.html`.

4 CASE STUDY IN UML

Class diagrams are used in UML for modeling classes and their relationships. Furthermore, the popular use of class diagrams is mainly to model the *domain* of software systems.

As structural models, presentation models can be built using class diagrams, as presented in Figure 2. There is modeled the presentation of the UI required to support the `Connect` functionality in Figure 1. Unlike in the modeling of the domain model, the use of class diagram for presentation modeling is not a popular choice. Indeed, it is difficult to realize that the class diagram in Figure 2 represents the presentation of a user interface, even when many important properties such as interaction object containments in a UI are actually modeled. However, this kind of abstract UI presentation is conceptually equivalent to the abstract presentation in MB-UIDEs [4, 8]. Further, the `InteractionClass`, `InvokeActionClass` and `PrimitiveInteractiveClass` classes in Figure 2 specify methods similar to those presented by Holub [9], a long-term practitioner working with UI design.

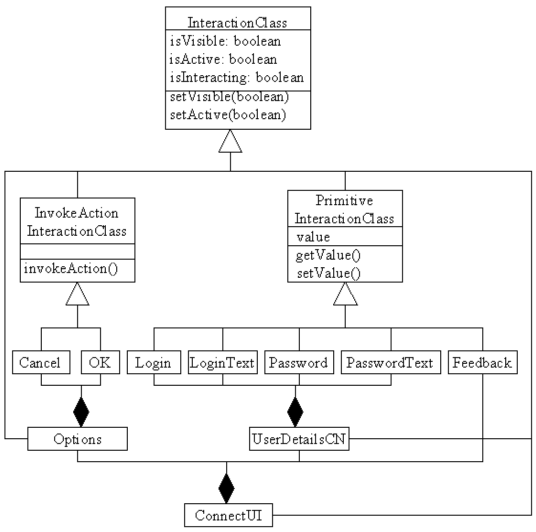

Figure 2: The `ConnectUI` presentation modeled in a UML class diagram.

Behavior can be modeled in UML using activity diagrams and interaction diagrams (i.e., collaboration diagram and sequence diagram). Interaction diagrams are more popular than activity diagrams for modeling behavior and specially for eliciting structural elements required to support a specific functionality. However, interaction diagrams suffer from difficulties when representing the non-sequential behaviors commonly required to model UI behaviors. In this case, activity diagrams as presented in Figure 3 can be used to describe the possible behavior of a user interface. The `Connect` activity in Figure 3 represents the behavior associated with the `Connect` use case in Figure 1. Therefore, for example, the `cn1` object of class `Cancel` is made active and awaiting for an action invocation immediately after the widgets in the `ConnectUI` are instantiated and made visible in the `InitiateConnectUI` activity. Further, the `uq` object of class `UserQuery` is instantiated by the `new UserQuery` action state leading to the execution of the `GetUserDetails` activity, all of these actions happening in parallel with the activation of the `cn1` object.

Although uncommon, this approach of using activity diagrams for modeling UI behavior is observed in UI designs produced by practitioners [10]. Furthermore, recalling the design guidelines of Shneiderman [22], activity diagrams allow designers to verify if UIs can *offer informative feedback*, if their sequence of actions are logically grouped in order to *yield closure*, and if they can *permit easy reversal of actions*. For instance, in Figure 3 it is specified that a user interacting with the `Connect` service can either gain access into the library system or receive "Invalid Information" feedback. Further, the `ok1` object is designed to be the last action in the `Connect` service, which provides a sense of service closure. Finally, the `cn1` object allows users to cancel an attempt to connect to the library system when the `ConnectUI` is visible.

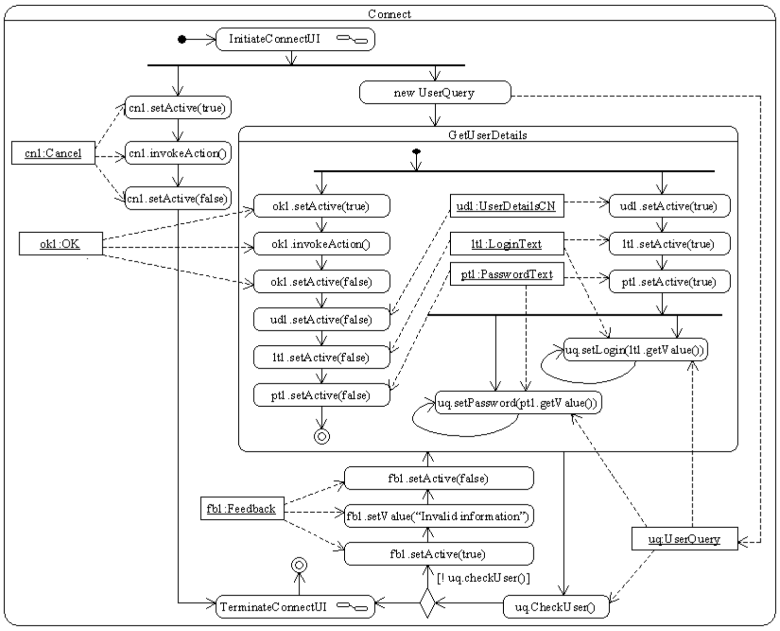

Figure 3: The behavior associated with the `ConnectUI` user interface modeled in a UML activity diagram.

Another benefit of activity diagrams is that they provide an explicit way to represent the complex relationship between structural and behavioral diagrams. Indeed, with the use of object flows as represented in Figure 3, activity diagrams can model the data flow in addition to the control flow of an interactive system.

As stated earlier, there are many different ways of modeling an interactive system. The UML model presented here is one approach which conforms with modeling techniques suggested by practitioners [9, 10] as well as with design guidelines suggested by HCI experts [4, 22].

5 CASE STUDY IN UML*i*

Some of the limitations of UML, such as the difficulty of visualizing widget containment in Figure 2, are apparent. To cope with these limitations, UML*i* proposes some extensions to UML. The *user interface diagram*, a specialized version of the class diagram, aims to support the design of UI presentations. Figure 4 presents a user interface diagram for the presentation modeled in Figure 2. There, the dashed cube, , is a `FreeContainer` representing a top-level interaction object which cannot be contained by any other interaction object, e.g. a top-level window. The dashed cylinders, are `Containers`, which can group interaction objects that are not FreeContainers. The downward triangles, ∇, are `Inputters`, which are responsible for receiving information from users. The upward triangles, $\triangle$, are `Displayers`, which are responsible for sending visual information to users. The pairs of semi-overlapped triangles pointing to the right, are `ActionInvokers`, which are responsible for receiving information from users in the form of events. Essentially, the user interface diagram notation provides two benefits when compared with class diagrams. Firstly, it emphasizes the notion of containment among interaction objects. Secondly, it identifies the abstract roles that widgets are playing in a user interface presentation [4].

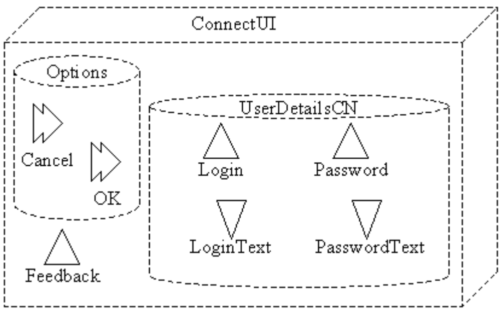

Figure 4: The `ConnectUI` presentation modeled in a UML*i* user interface diagram.

The modeling of UI behavior is simplified in UML*i* by extending the notation of activity diagrams, as presented in Figure 5. For example, the `order independent` Selection State, $\ominus$, simplifies the modeling of a state where users have a range of choices (or *selectable states*) that can be performed however many times required, if any. Then, the action states `uq.setLogin(getValue())` and `uq.setPassword(getValue())` are the selectable states for the selection state in Figure 5. Further, these selectable states correspond to the `uq.setLogin(lt1.getValue())` and `uq.set Password(pt1.getValue())` action states respectively in Figure 3. Moreover, the pattern of action states, transitions and forks within the `GetUserDetails` in Figure 3 corresponds to the `order independent` Selection State in Figure 5. Finally, there is a set of powerful stereotypes, e.g., ≪*presents*≫, ≪*cancels*≫, ≪*interacts*≫, and ≪*confirms*≫, based on the use of interaction object flows that reduce the necessity of modeling action states related to the process of making widgets visible, invisible, active and inactive. A description of these stereotypes in terms of UML constructors is presented later in the paper.

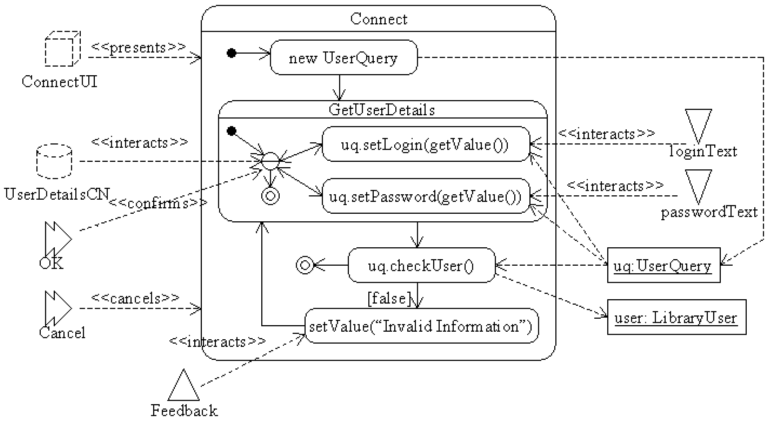

Figure 5: The behavior associated with the `ConnectUI` user interface modeled in a UML*i* activity diagram.

6 DESIGN METRICS

This section describes the metrics selected to assess the models produced. It also describes the process of translating UML*i* into UML in order to produce a pair of models specifying an *equivalent* set of properties of the library system.

6.1 Selected Metrics

Structure, behavior and visual appearance are the three dimensions of interactive system designs considered in this case study.

Structural Complexity. CK metrics [6] is considered in this paper for measuring structural complexity. However, from the CK metrics, only the CBO and RFC metrics, defined as follows, are mentioned in this paper. In fact, according to the study in this paper, the other CK metrics were not significantly affected when modeling UIs using standard UML.

- *Coupling Between Object Classes (CBO)* is defined as the number of classes to which a class is coupled. This metric measures the complexity of modifying and testing a class in relation to other classes. The assumption regarding this metric is that highly coupled classes are more fault-prone than weakly coupled classes.
- *Response For a Class (RFC)* is defined as the number of methods that can be executed in response to a message received by an object of that class. This metric measures the number of activities and action states reached by transitions triggered by operations of a class. The assumption regarding this metric is that the larger the response set of a class, the higher is the complexity of the class, and consequently the more fault-prone and difficult to modify is the class.

Behavioral Complexity. McCabe's cyclomatic complexity (CC) metric [14] is defined as the number of decisions (or predicates) specified in models, plus 1. Since decisions are specified in behavioral models, this is a metric for behavioral complexity. The reasoning behind this metric is that CC corresponds to the number of possible execution paths specified in the models. The assumption regarding this metric is that models with high CC are more difficult to understand and consequently maintain than models with low CC.

Visual Complexity. There is no well-established metric for measuring the complexity of visual languages [7]. In the case of Argo/UML*i*, the diagrams can be stored and exchanged using the Precision Graphics Markup Language (PGML) format. Therefore, as PGML is a textual representation for the diagrams in Argo/UML*i*, the number of lines of code (LOC) of the PGML files is the metric for measuring the size of the UML and UML*i* diagrams in this study. Size, however, may not be an appropriate metric for visual complexity since designs with long textual representations can be very simple ones. Therefore, the following metrics are used in this paper for measuring the visual complexity of the models.

- *Density of coupling between objects in diagrams (DCBOD).* This is defined as the ratio of the level of CBO in the models and the total number of LOC of the PGML files representing the diagrams. The non-validated assumption is that high densities indicate that more relevant structural specification, e.g., structural complexity, can be represented by fewer graphical elements than with low densities.

- *Density of cyclomatic complexity in diagrams (DCCD).* This is defined as the ratio of the cyclomatic complexity in the models and the total number of LOC of the PGML files representing the diagrams. The non-validated assumption is that high densities indicate that more relevant behavioral specification, e.g., behavioral complexity, can be represented by fewer graphical elements than with low densities.

6.2 Using the Design Metrics

Measuring the metrics in the models of the library system is a straightforward task. The major concern regarding the use of these design metrics is the production of models using two different notations to model a common set of properties from the specification of a system. Therefore, a systematic mapping strategy should be defined in order to ensure that the same reuse strategy is used in both models. Different reuse strategies can significantly affect metrics [2].

The diagrams in Figures 4 and 5 correspond to the modeling of the `Connect` use case in Figure 1 using UML*i*, while the diagrams in Figures 2 and 3 correspond to the systematic translation of the UML*i* models into UML models. Therefore, the mapping rules from UML*i* into UML can be explained with reference to the Figures 2, 3, 4 and 5.

InteractionClasses to Classes. InteractionClass, InvokeActionInteractionClass and PrimitiveInteractionClass in Figure 2 are classes of the UML*i* metamodel used to represent the InteractionClasses, viz., FreeContainers, Containers, Inputters, Displayers, Editors and ActionInvokers. Therefore, the InteractionClasses in Figure 4 are mapped into Classes in Figure 2. This is a natural mapping since the UML*i* InteractionClass is a subclass of UML Class. As a consequence of this mapping we can observe the following.

- *Placement to Composition*. In UML*i*, an Interaction-Object which is not a FreeContainer must be associated with a Container. Thus, the association of an *InteractionObject* to its Container is specified in UML*i* by the placement of the InteractionObject into the Container in the diagram. In UML, however, this association is specified by a composition. Therefore, a composition is created from each InteractionClass in Figure 2 that is not a FreeContainer to its immediate Container. For instance, Cancel is placed in Options in Figure 4, so there is a composition between Cancel and Options in Figure 2.
- *Visible(), Active() and InvokeAction() operations to explicit class methods*. These operations originally embedded in the UML*i* metamodel are explicitly modeled in the UML presentation model in Figure 2.

Interaction Object Flow to Object Flow. Interaction object flows in Figure 5 were translated into object flows in Figure 3. Despite missing the information as to which role each object can play in a UI, this is a natural mapping since InteractionObject is a subclass of Object in the UML*i* metamodel.

Interaction object flow stereotypes into fragments of a standard activity diagram. There is no one-to-one mapping between UML*i* constructors and UML constructors. Thus, the mapping of each stereotype must be individually explained.

- The ≪*Presents*≫ stereotype in Figure 5 specifies that the ConnectUI FreeContainer is a presentation unit. This means that the widgets directly and indirectly contained by ConnectUI must be instantiated (if not previously explicitly instantiated) and must be made visible when the Connect activity is reached as specified in UML by the InitiateConnectUI activity in Figure 6. Further, these widgets must be made invisible and destroyed when the Connect activity is left as specified in UML by the TerminateConnectUI activity also in Figure 6.
- The ≪*Cancels*≫ stereotype in Figure 5 specifies that the Cancel ActionInvoker is active and can finish the Connect activity anytime when the controlflow is there. This behavior is modeled in Figure 3 by the Cancel object that is made active immediately after the instantiation of ConnectUI.

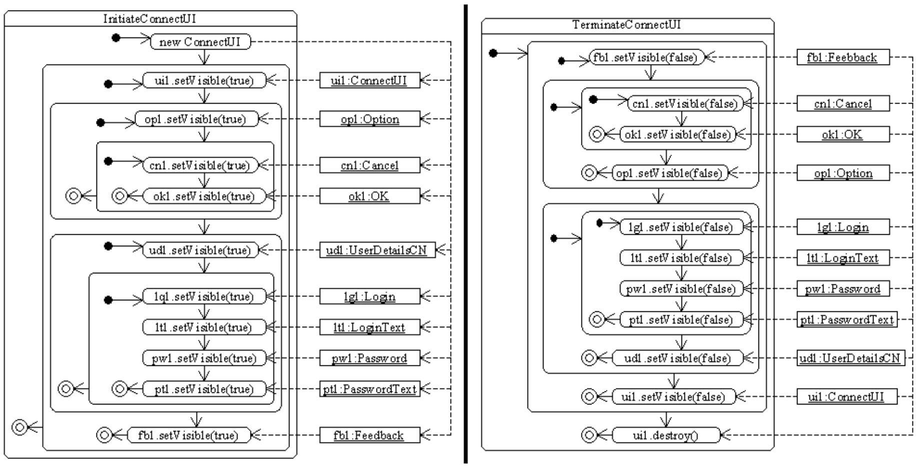

Figure 6: UML mapping of UML*i* ≪*Presents*≫.

- The ≪*Interacts*≫ stereotype can be associated with a `Container` or `PrimitiveInteractionObject`. If the stereotype is associated with a `Container`, this means that contained interaction objects are made active when the associated activity or action state is reached. If associated to a `PrimitiveInteractionObject`, this means that the object is made active (if its `Container` was not previously activated by another ≪*Interacts*≫) and ready to interact with a user through the `getValue()` and `setValue()` operations.

- The `OrderIndependent` SelectionState in Figure 5 corresponds to the fork with each selectable action being recursively invoked, as presented in Figure 3. Thus, the ≪*Confirms*≫ stereotype in Figure 5 is mapped into three sequential action states associated to the `OK` object in Figure 3. The first action state activates the associated `ActionInvoker`, the second waits for the performance of the `invokeAction()`, and the last deactivates the `ActionInvoker`. Therefore, the performance of the `invokeAction()` is responsible for *confirming* the end of the selection state in Figure 3.

- The ≪*Activates*≫ stereotype, not used in Figure 5, is mapped into three sequential action states, as in ≪*Confirms*≫. The important characteristic of ≪*Activates*≫, though, is that these sequential action states are placed before the activity/action state to be triggered by the associated ActionInvoker.

7 RESULTS

Two set of files containing a textual representation of the UML and UML*i* models were produced using Argo/UML*i*. Table 1 presents the consolidated size in terms of LOC of these sets. There, the size of the textual representations of the models and diagrams are 3.44 times higher and 4.01 times higher in UML than in UML*i* respectively.

Model	LOC of XMI	LOC of PGML
UML	58,029	125,658
UML*i*	16,836	31,325
UML/UML*i*	3.44672	4.01047

Table 1: Size of the UML and UML*i* models of the library system.

Size usually does not say much about how difficult it is to construct model of an interactive system or how difficult it is to understand such models. By contrast, the metrics in this paper are designed to quantify the inherent difficulties of constructing and understanding models, since they measure many dimensions of the complexity associated with the models. Table 2 presents the metrics for the UML and UML*i* models. The following results can be observed analyzing Table 2.

Model	Structural Complexity			Behav. Cmplx.	Visual Complexity	
	Data	CBO	RFC	CC	DCBOD	DCCD
UML	Mean	2.7582	6.7582	77	0.0019	0.0006
	Sum	251	615			
UML*i*	Mean	2.8523	1.1818	66	0.0080	0.0021
	Sum	251	104			
UML/ UML*i*	Mean	0.9670	5.7185	1.1667	0.2484	0.2891
	Sum	1.0	5.9135			

Table 2: Design metrics of the UML and UML*i* models of the library system.

- The total of CBO is same in both models. The difference in the mean is due to the abstract `InteractionClass`, `InvokeActionInteractionClass` and `PrimitiveInteractionClass` classes which are explicitly specified in the UML models and implicitly specified in the UML*i* metamodel. This means that CBO is not affected at all by UML*i* even with a model corresponding to 29% the size of the UML model as indicated in Table 1.

- RFC has been reduced 4.88 times when using UML*i* since much of the behavior in UML*i* classes is embedded within the UML*i* constructs, making the models more straightforward and easier to maintain. As a result of such improvement, Figure 7 presents a graphical representation of the distribution of RFC per number of classes. In this distribution, small areas are better than big areas since low RFCs are better than high RFCs. Further, according to [6], the reduction of RFC indicates that UML*i* classes are less fault-prone and easier to maintain than UML classes. Therefore, *the reduction in RFC is a significant achievement of UMLi*.

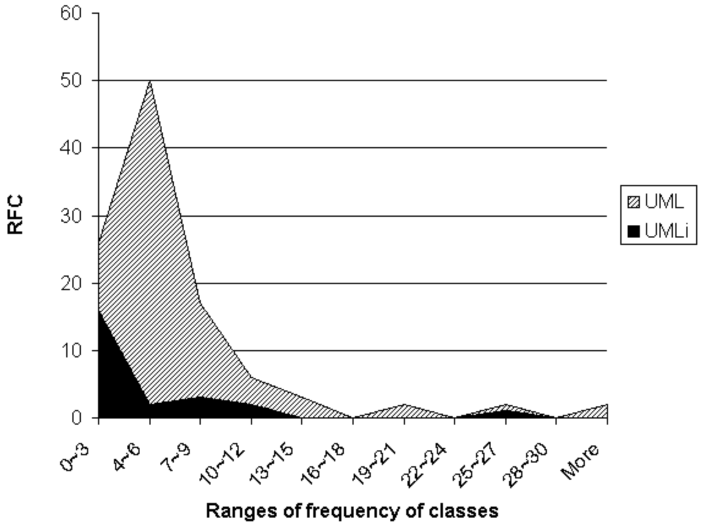

Figure 7: Distribution histogram comparing the RFC of the UML and UML*i* models.

- CC is 16% higher in UML than in UML*i*. This is due to the ≪*cancels*≫ and ≪*confirms*≫ stereotypes that eliminate the necessity of specifying how activities can be canceled by users and how *option selection states* can be confirmed by users. This reduction may have a significant impact in the design of large-scale interactive systems since it provides an uniform treatment for the canceling of tasks in order to allow users to reverse their actions [22].
- DCBOD is 3 times lower and DCCD is 2.45 times higher in UML than in UML*i*. This indicates that the capability of UML*i* to visually represent both structural and behavioral complexity is higher than that of UML. Thus, with a fixed number of graphical elements, UML*i* should be able to represent more relevant information than UML.

From a metric point of view there is no trade-off involved in the use of UML*i* compared with UML for modeling interactive systems. In fact, no metric has demonstrated any disadvantage to the use of UML*i* for modeling interactive systems when compared with UML.

8 CONCLUSIONS

This study has demonstrated in a quantitative way that the UML*i* proposal [20] is an improvement on UML for UI design. Further, it has demonstrated that UML can usefully be enhanced for modeling UIs, as qualitatively identified in studies on the use of UML for modeling UIs [11, 18]. In fact, significant reductions in structural complexity (reduction of 87% in RFC) and visual complexity (increase of 402% in DCBOD and 345% in DCCD) along with considerable reductions in behavioral complexity (reduction of 14% in cyclomatic complexity) are achieved in UML*i* models by improving the support of UML for UI design. These metric improvements, according to their validated assumptions, mean that the construction and maintenance of models of interactive systems should be simpler and easier in UML*i* than in UML. Indeed, the introduction of the user interface diagram with its interaction classes has simplified the visualization of UI presentations. The introduction of the interaction object flow with its stereotypes has simplified the modeling of actions related to UI widgets. The introduction of selection states has simplified the modeling of behaviors usually observed in UIs.

As well as providing a systematic comparison of UML and UML*i*, the paper establishes a foundation for a productive discussion on how to assess UML extensions for UI design using design metrics. Any other proposal to extend UML for UI design, e.g., [13], [15] and [18], can be contrasted, for example, with UML*i*, by comparing the metrics presented with those for standard UML. This approach would allow researchers to systematically identify the best combination of improvements in UML to support UI design.

Finally, this paper has demonstrated that well established object-oriented design metrics can contribute to the evaluation of user interface designs early in the process of developing UI software. For instance, using design metrics, UI models in UIMSs and MB-UIDEs could be evaluated before any running UI is produced from them. Indeed, user interface design attributes can be directly evaluated from their internal attributes rather than from any other artifact, e.g., UI prototypes or UI code, generated from them.

References

[1] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Trans. Software Engineering*, 22(10):751–761, October 1996.

[2] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. How Reuse Influences Productivity in Object-Oriented Systems. *Communications of ACM*, 39(10):104–116, 1996.

[3] James M. Bieman and Byung-Kyoo Kang. Measuring Design-level Cohesion. *IEEE Trans. Software Engineering*, 24(2):111–124, February 1998.

[4] François Bodart and Jean Vanderdonckt. Widget Standardisation Through Abstract Interaction Objects. In *Advances in Applied Ergonomics*, pages 300–305, Istanbul - West Lafayette, May 1996. USA Publishing.

[5] Lionel C. Briand, Sandro Morasca, and Victor R. Basili. Defining and Validating Measures for Object-Based High-Level Design. *IEEE Trans. Software Engineering*, 25(5):722–743, September/October 1999.

[6] Shyam R. Chidamber and Chris F. Kemerer. A Metric Suite for Object Oriented Design. *IEEE Trans. Software Engineering*, 20(6):476–493, 1994.

[7] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thonson, London, UK, second edition, 1996.

[8] Tony Griffiths, Peter J. Barclay, Norman W. Paton, Jo McKirdy, Jessie B. Kennedy, Philip D. Gray, Richard Cooper, Carole A. Goble, and Paulo Pinheiro da Silva. Teallach: A Model-Based User Interface Development Environment for Object Databases. *Interacting with Computers*, 14(1):31–68, December 2001.

[9] Allen I. Holub. Building User Interfaces for Object-Oriented Systems. *Java-World*, July-November 1999, January 2000, March 2000.

[10] Ari Jaaksi, Juha-Markus Aalto, Ari Aalto, and Kimmo Vättö. *Tried & True Object Development: Practical Approaches with UML.* Cambrigdge University Press, Cambridge, UK, 1999.

[11] Srdjan Kovacevic. UML and User Interface Modeling. In *Proceedings of UML'98*, pages 235–244, Mulhouse, France, June 1998. ESSAIM.

[12] Wei Li and Sallie M. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.

[13] Panos Markopoulos and Peter Marijnissen. UML as a representation for Interaction Designs. In *Proceedings of OZCHI 2000*, pages 240–249, 2000.

[14] Thomas J. McCabe and Charles W. Butler. Design Complexity Measurement and Testing. *Communications of the ACM*, 32(12):1415–1425, December 1989.

[15] Nuno J. Nunes and João Falcão e Cunha. Towards a UML profile for interaction design: the Wisdom approach. In *Proceeding of UML2000*, volume 1939 of *LNCS*, pages 101–116, York, UK, 2000. Springer.

[16] Object Management Group. *OMG Unified Modeling Language Specification*, June 1999. Version 1.3.

[17] Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer, Berlin, 1999.

[18] Fabio Paternò. Towards a UML for Interactive Systems. In *Proceedings of EHCI2001*, LNCS, pages 7–18, Toronto, Canada, May 2001. Springer.

[19] Paulo Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. In *Proceedings of DSV-IS2000*, volume 1946 of *LNCS*, pages 207–226, Limerick, Ireland, June 2000. Springer-Verlag.

[20] Paulo Pinheiro da Silva and Norman W. Paton. UMLi: The Unified Modeling Language for Interactive Applications. In *Proceedings of UML2000*, volume 1939 of *LNCS*, pages 117–132, York, UK, October 2000. Springer.

[21] Paulo Pinheiro da Silva and Norman W. Paton. A UML-Based Design Environment for Interactive Applications. In *Proceedings of UIDIS'01*, pages 60–71, Zurich, Switzerland, May 2001. IEEE Computer Society.

[22] Ben Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, MA, third edition, 1997.

[23] Pedro A. Szekely, Piyawadee Noi Sukaviriya, Pablo Castells, Jeyakumar Muthukumarasamy, and Ewald Salcher. Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. In *Engineering for Human-Computer Interaction*, pages 120–150, London, UK, 1996. Chapman & Hall.