TUIMS: Laying the Foundations for a Tangible User Interface Management System

Orit Shaer, Robert J.K Jacob
Computer Science Department
Tufts University
Medford, MA 02155, USA
E-mail: oshaer@cs.tufts.edu

ABSTRACT

The development of Tangible User Interfaces entails a unique set of challenges in comparison to traditional GUI development. These challenges range from conceptual through methodological to technical. To address these challenges, this paper proposes a new class of software tools for TUIs: the Tangible User Interface Management System (TUIMS). TUIMS draws from earlier work on UIMS [8] and provides an alternative paradigm for developing TUIs. Rather than developing a TUI using an API for a specific sensing mechanism, TUI developers would specify a TUI in a specialized high level description language which is technologically independent. This technology independent specification can then be translated into a program controlling a set of physical objects in a specific target technology. To demonstrate the benefits of a TUIMS, we are currently developing Prism. Prism is a TUIMS aimed at: reducing the effort required to build a TUI, encouraging exploration of alternative designs, and lowering the threshold for retargeting a TUI to new input/output technologies. The Prism system will support development of TUIs based on microcontrollers and RFID readers. Prior to developing a TUIMS, it was necessary to identify the set of high level constructs that would serve as a basis for a high level description language for TUIs, in the same way that widget, windows and events formed the basic ingredients of GUI toolkits. Therefore, we proposed the TAC Paradigm [10], a conceptual framework for TUIs providing a set of core constructs for describing TUIs while addressing many of the conceptual challenges unique to the rich design space of TUIs. By providing a simple set of constructs, the TAC paradigm enables construction of a high level description language for TUIs, thus laying the foundations for the development of a TUIMS.

KEYWORDS

Tangible User Interface (TUI), User Interface Management System (UIMS), User Interface Description Language (UIDL), Software toolkit.

INTRODUCTION

The last decade has seen a wave of new research aimed at fusing the physical and digital worlds. This work has led to the development of a collection of interfaces allowing users to take advantage of their spatial skills and to interact collaboratively with augmented physical objects in order to access and manipulate digital information. These interfaces are referred to as Tangible User Interfaces [3] (TUIs). Interaction with TUIs draws on users' existing skills of interaction with the real world, thereby offering the promise of interfaces that are quicker to learn and easier to use. However, these interfaces are currently more challenging to build than traditional user interfaces. Following are a number of conceptual, methodological and technical challenges TUI developers face. A more comprehensive discussion of these challenges can be found in [10].

Interlinked Virtual and Physical Worlds: While graphical user interfaces rely solely on virtual objects, Tangible user interfaces make use of both virtual and physical objects, which coexist and exchange information with each other. The TUI developer is challenged with determining which information is best represented digitally and which is best represented physically [11].

Continuous and Distributed Interaction: TUIs provide users a set of physical objects with which they can interact in a discrete or continuous fashion. In addition, multiple users can simultaneously interact with multiple physical objects. Existing user interface models such as Event Based models do not capture continuous and discrete interaction explicitly [4], thus, TUI developers are often required to deal with continuous and distributed interaction in considerably ad-hoc, low-level programming approaches.

*Multiple behaviors, objects and actions***:** In existing user interface paradigms each interactive component encapsulates its behavior. However, the behavior of a physical object in a TUI may change in different contexts of use (e.g. when a new physical object is added to the TUI the behavior of an existing object may change). Furthermore, in a TUI, any object that a user may grab from her physical surroundings could be part of an interface and there are numerous interaction actions that can be performed with, or on any physical object (e.g. squeeze, stroke, toss, push, tap, pat, etc.). Thus the existing user interface paradigms fall short of capturing the dynamic behavior of TUIs.

No Standard Input / Output devices: Currently there are no standard input or output devices for accomplishing a given task in a TUI (e.g. measuring a movement of an object can be implemented using RFID, magnet sensation or computer vision). Because each technology currently requires a different set of physical devices and code instructions, the integration of novel technologies into an application is difficult as well as costly [5].

Early Feedback: Unlike homogeneous desktop systems having standard and available input/output devices; TUIs use novel hardware that may not be available early in the design process. Thus, a rapid prototype to simulate the functionality and the hardware is needed [7] to avoid postponing testing until the whole system has been developed. However, building a proof of concept prototype using available technology may require rewriting the TUI software when the actual deployment technology is selected.

To address these challenges, we suggest a new class of software tools for TUIs: the Tangible User Interface Management System (TUIMS). Unlike existing physical toolkits which provide support and abstractions for a specific set of sensing mechanisms, the TUIMS provides a higher level model that is aimed at capturing the essence of the tangible interaction. The TUIMS, which draws from earlier work on UIMS [8], allows developers to easily specify a TUI using a specialized high level description language (TUIDL). This technology independent specification can then be translated into a program controlling a set of physical objects in a specific target technology.

In order to provide a basis for a high level description language for TUIs, we proposed the TAC Paradigm [10]. The TAC paradigm is a conceptual framework for TUIs, providing core constructs for describing these systems which are analogues to widgets, windows and events in GUIs.

To demonstrate the benefits of a TUIMS to developers, Prism is currently being developed. Prism is a TUIMS instantiation aimed at enabling TUI developers with limited experience in hardware programming to accomplish the following:

1. Implement the core functionality of a TUI while relying on the Prism system to implement the communication with the input/output devices, thus reducing the effort required to program a TUI system.
2. Create a model which describes the structure and behavior of a TUI in a high level description language. Since the model is technology independent, it can be easily retargeted to support other input/output technologies (e.g. microcontrollers, RFID and 3D graphical simulation).
3. Explore alternative designs (i.e. different physical representations) for the same application functionality, while reusing the original application logic. This reduces the time necessary to build different systems with similar functionality.
4. Rapidly prototype a TUI from a high level description early in the development process using a 3D graphical simulation environment or available hardware, thus, providing early feedback to the system developer. The high level description used for the early prototype can then be incrementally developed to create the deployable system.

The Prism system is intended to enable prototyping and development of TUIs, while providing support for the following input/output technologies: a 3D graphical simulation environment, the HandyBoard and Cricket microcontrollers, as well as an RFID reader. The system will enable TUI developers to augment TUI's with speech, sound and video streams. Its flexible architecture will allow developers to easily extend Prism to support additional technologies.

The following sections are organized as follows: First the TAC Paradigm, laying the conceptual foundations for TUIMSs is summarized. Next follows a discussion of how to model TUIs using the TUIML high level description language. Finally, the TUIMS architecture is presented and is demonstrated through the Prism system. The paper closes with a summary and our future plans.

THE TAC PARADIGM

The Token and Constraints (TAC) Paradigm [10] provides a set of constructs for describing TUIs while addressing many of the conceptual challenges such as *multiple behaviors, objects and actions* as well as *continuous and distributed interaction*, unique to the rich design space of TUIs. By providing a set of constructs for TUIs which are equivalent to widgets, windows and events for GUIs, the TAC paradigm lays the foundation for a high level description language for TUIs.

The TAC paradigm approach is based on describing a TUI as a set of relationships between two types of physical objects: *tokens* and *constraints*. A *Token* is a physical object that represents digital information or a computational function and is manipulated by the user in order to access or modify digital information. A *Constraint* is a physical object that provides the context for *token* manipulation. The relationship between a *token* and a set of *constraints* is called a *TAC*. Each *TAC* encapsulates a set of manipulation actions that can be performed on it. By encapsulating the manipulation actions in the *TAC* relationship rather than in the *token* entity, the paradigm allows a different behavior to be specified for each token in each different context, thus resolving the challenge of *multiple behaviors*.

The manipulation of a *TAC*, is considered the manipulation of a *token* in respect to a set of *constraints.* The manipulation has computational interpretation. For example, in the Marble Answering Machine [2], we consider a marble the *token* and the replay indentation the *constraint*. The manipulation of the marble in respect to the replay indentation has computational interpretation: it provides access to the message bound to the marble. Each TAC can be manipulated discretely, continuously or in both ways.

To specify a TUI using the TAC paradigm, a developer defines the possible *TACs* within a TUI and describes their behavior. For each *TAC,* the developer specifies the actions that may be performed upon it, and their responses. These relationships may be instantiated at run time by either the user or the system. The simplicity of specifying a TUI using the TAC paradigm implicitly supports distributed interaction by maintaining a set of parallel *TAC* relationships.

Thus far, we have discussed the TAC Paradigm and identified TACs as the conceptual building blocks of TUIs. Similar to widgets in a GUI, TAC objects encapsulate the set of manipulation actions that can be performed upon a physical object in a TUI. A detailed discussion of the TAC Paradigm terminology, properties and evaluation can be found in [10].

MODELING TANGIBLE USER INTERFACES

Building upon the set of constructs provided by the TAC paradigm we introduce TUIML (Tangible User Interface Markup Language), a high level description language for TUIs. The TUIML design, which is influenced by recent user interface description languages [1,4,9], provides support for the entire life cycle of a TUI while using XML as an underlying technology.

TUIML predefines five basic modules: *Task, Domain, Representation, TAC* and *Control,* each describing a different aspect of the TUI system**.**

The *task* and *domain* modules describe the semantics of the TUI system. The *representation* and *TAC* modules describe the syntax of the TUI system. The *representation* module defines a set of logical physical objects and their properties. The *TAC* component defines the context for interaction actions performed upon these logical physical objects and determines which semantic functions are invoked as a result of an interaction action. The *control* component describes the control flow of the tangible interaction and the TUI's initial state. These are modeled using a Colored Petri Nets (CPN) [6] based formalism that captures both the parallel and the continuous characteristics of a TUI

In order to validate the expressiveness and usefulness of TUIML, we are currently taking a number of validation activities including hand coded representations of new and existing TUIs. Having validated the language, it will be used to specify TUIs in our proposed TUIMS system.

TUIMS ARCHITECTURE

The design of the TUIMS architecture is intended to facilitate the development of technologically portable TUI systems and allow TUI developers to extend the number of input and output technologies supported by the TUIMS. This architecture, which draws from the UIMS architecture [8], is depicted in figure 1.

The main components of the architecture are: the *modeling tools,* the *model,* the *implementation tools,* the *dialogue manager* and the *lexical handlers*.

The *model* is the core component of the system. It organizes the information into the five TUIML modules: *task, domain, representation, TAC* and *control***.** Its modular structure allows incremental development and reuse of existing modules throughout the development of an

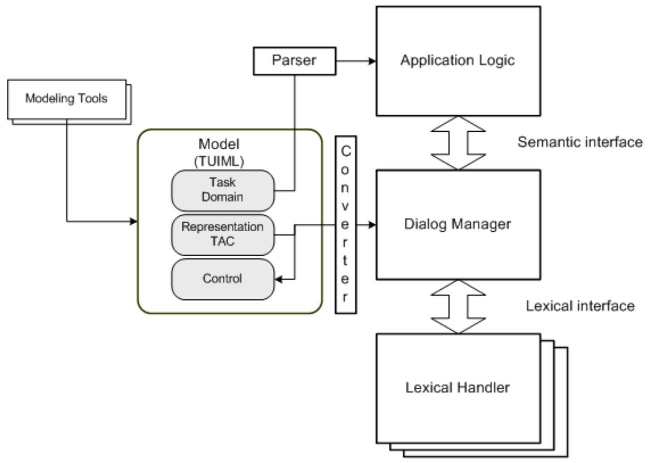


Figure 1: The TUIMS architecture

individual TUI system and across the development of different TUI systems.

The *modeling tools* are aimed at assisting developers in building the *model* while shielding them from the syntax of the modeling language. They provide a convenient way to specify the interface and access existing specifications.

The *implementation tools* translate the TUIML specification into programming language source code, thus assisting the TUI developer in the implementation process.

Lexical Handlers are programs that control the communication and user interaction with a given sensing mechanism. Each *lexical handler* is responsible for both establishing communication with the set of devices included in a given sensing mechanism, and for generating events to and from those devices. Events generated by a *lexical handler* are sent to the *dialogue manager*. Each event sent contains an ID indicating a specific physical object and other necessary information. In order to extend TUIMS support to a new sensing mechanism, the TUI developer simply needs to provide a lexical handler for the new mechanism.

The *Dialog Manager* is driven from the *representation, TAC* and *control* modules. It is responsible for both binding actual or graphical physical objects to the logical physical objects defined in the representation module, and for invoking application functions in response to events received from the lexical handlers. The dialogue manager keeps track of the instantiated physical objects and their IDs. It contains an entry for each lexical handler that maps input/output events from the lexical handler to the syntactic interaction actions. By modifying these entries a TUI developer can easily retarget a TUI or parts of a TUI to a different sensing mechanism. When a new physical object is initialized, its lexical handler sends its ID to the dialog manager which in turn, instantiates a new logical physical object and stores an entry mapping the physical object ID to the logical physical object ID. Any manipulation action performed upon a physical object or a change in the state of a logical physical object is reported to the dialog manager; in response the dialog manager invokes the proper function from the application logic or updates the relevant lexical handler.

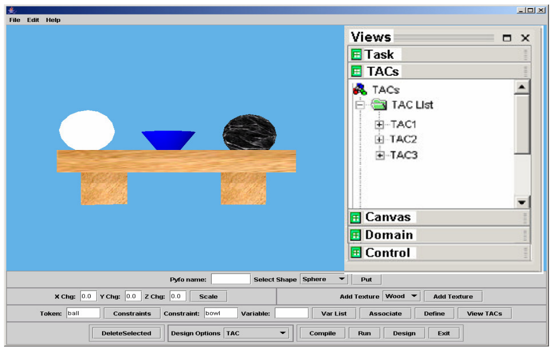
PRISM

The Prism environment employs the TUIMS architecture and is implemented in Java using the Java3D, Java Media Framework and Java Speech APIs. It allows a TUI developer to specify a TUI in the TUIML description language. This specification can then be automatically translated into a graphical simulator or a program controlling a set of physical interaction objects.

The Prism environment aims to be both a TUI development environment and a run-time environment, where run-time services can be provided.

The development environment consists of several views, each used as a modeling tool for specifying a different aspect of the TUI. These specifications are saved as a TUIML file and can be reloaded and reused. Prism provides the following views: *Task, Domain, Canvas, TACs* and *Control*. The *task* and *domain* views provide form based tools for specifying and implementing the application logic. The *canvas* view (see figure 2) provides users with a graphical editor for sketching 3D representations of physical objects. It's used for specifying the *representation* component by creating graphical representations of physical objects, then relating them to domain elements and combining them into TACs. The *canvas* is also used for simulating user interaction with the sketched objects and testing alternative representations of the same underlying functionality. The *TAC* view allows users to define manipulation actions for each TAC and relate these manipulation actions to elements from the *task* model. The *control* view provides a visual editor for specifying the control flow of the TUI and the TUI's initial state. It provides users a list of available *lexical handlers* and allows them to map low level events to TAC events.

The run-time environment will generate a *dialog manager* component from the TUI specification that will manage binding the semantic events to input/output events. It will provide run time support for a Java3D simulation environment, the HandyBoard and Cricket microcontrollers as well as an RFID reader. The environment will also generate source code from the specifications to be loaded into the HandyBoard and Crickets microcontrollers.



Select Shape: Sphere
X Chg: 0.0 Y Chg: 0.0 Z Chg: 0.0
Add Texture: Wood
Token: ball
Constraint: bowl
Variable:
Design Options: TAC

Figure 2: The Prism canvas view provides users a graphical editor for sketching and simulating 3D representations of physical objects.

SUMMARY AND FUTURE WORK

Our research is aimed at providing a new class of software tools for TUIs: TUIMS. These software tools rely on rich high level representations of TUIs, which capture both the structure and dynamic behavior of TUIs. We believe that a TUIMS will reduce the effort required to create a TUI, will encourage experimentation with alternative designs and will produce more reliable and technologically portable systems. To demonstrate the benefits of a TUIMS, we are currently building Prism, a TUIMS which supports 3D graphical simulation, rapid prototyping and development of TUIs using microcontrollers as well as an RFID reader. We intend to evaluate Prism by incorporating it into a TUI design class and using it to build new TUI systems, as well as to rebuild existing systems.

REFERENCES

1. Ali, M.F., et al. *Building multiplatform user interfaces using UIML*. in *Computer Aided Design of User Interfaces (CADUI'02)*. 2002.
2. Crampton Smith, G. *The Hand That Rocks the Cradle*. in ID magazine, May/June 1995, pp 60-65.
3. Ishii, H. and B. Ullmer. *Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms*. in *Proceedings of the ACM Human Factors in Computing Systems (CHI 97)*. 1997. Atlanta, Georgia, USA.
4. Jacob, R.J.K., L. Deligiannidis, and S. Morrison, *A Software Model and Specification Language for Non-WIMP User Interfaces*. ACM Transactions on Computer-Human Interaction (TOCHI), 1999. **6**(1): pp. 1-46.
5. Klemmer, S.R. and J.A. Landay. *Papier-Mache: Toolkit Support for Tangible Input*. in *Human Factors in Computing Systems: CHI2004*. 2004. Vienna, Austria: CHI Letters.
6. Kristensen L. M., Christensen, S. and Jensen K., *The Practitioner's Guide to Coloured Petri Nets,* International Journal on Software Tools for Technology Transfer, 1998, 2: pp. 98-132.
7. Myers, B., S.E. Hudson, and R. Pausch, *Past, Present, and Future of User Interface Software Tools*. ACM Transactions on Computer-Human Interaction (TOCHI), 2000. 7(1).
8. Olsen, D.R., *User Interface Management Systems: Models and Algorithms*. 1992: Morgan Kaufmann.
9. Puerta, A. and J. Eisenstein, *XIML: A Multiple User Interface Representation Framework for Industry*, in *Multiple User Interfaces*, A. Seffah and H. Javahery, Editors. 2003, John Wiley and Sons.
10. Shaer, O., Leland N., E.H Calvillo-Gamez, and R.J.K. Jacob, *The TAC Paradigm: Specifying Tangible User Interfaces*. Personal and Ubiquitous Computing (in press), 2004.
11. Ullmer, B., *Tangible Interfaces for manipulating aggregates of digital information*. PhD thesis, Massachusetts Institute of Technology, 2002.