TUIML: A Visual Language for Modeling Tangible User Interfaces

TUIML is a visual modeling language for Tangible User Interfaces (TUIs) aimed at specifying, analyzing and refining tangible interaction. It consists of an interaction model and diagramming techniques for describing the structure and behavior of TUIs in a high-level technology independent manner.

Here, we explain how to describe the structure, functionality and behavior of TUIs using TUIML.

In order to clearly explain the TUIML model and notation, we have selected to use an existing TUI as a leading example throughout this paper. We have selected URP as our example interface because it is one of the most fully developed and widely known TUI systems. It also serves as a good example for a generic interactive surface TUI.

1. Describing the Structure of a TUI using TUIML

1.1 A set of constructs

Before a modeling language for TUIs can be defined, it is first necessary to identify the set of constructs required to describe the structure and functionality of a large subset of TUIs. Our TAC paradigm provides a set of core constructs, which are for a wide range of TUIs, what widgets and events are to GUIs. Here we present these constructs and introduce a visual notation to represent them.

Our approach is based on the notion that the structure of a TUI can be described as **a set of relationships between physical objects and digital information.** Such relationships are defined by the TUI developer at design time and are actually instantiated at run time, either by the system or by the user. After a relationship has been instantiated, a user may interact with physical objects in order to access or manipulate the digital information they represent.

We would like to begin by presenting the following constructs: *Token*, *Constraint*, and *TAC*. For each of these constructs we provide a definition and a visual notation that represent it. After presenting these constructs we show how to use them to describe TUIs. The Urp system will be used as an example throughout both the introduction of the TUIML constructs and the ensuing discussion of the TUIML model and notation.

A Token is **a graspable physical object that represents digital information or a computational function** in an application. In other words, a token is a physical object that is binded to an application variable. While it is the role of the TUI designer to define what type of variable may be binded to a certain physical object. The actual binding of a variable to a token could then be performed either by the designer at design time or by the user at run-time. A physical object is considered a token only after it is binded to a variable.

Users interact with tokens in order to access or manipulate the digital information they represent. The physical properties of a token (e.g. size, texture, color etc.) may reflect the nature of either the information or the function it represents as well as suggest how it is to be manipulated. The physical properties of a token could be computationally augmented to provide users with haptic, visual or audible feedback. We refer to tokens that provide users with such augmented feedback as active tokens. One of the TUI that uses active tokens is the TVE.

For example, we consider the building models in the Urp system as tokens because each physical building model represents a virtual building in a computational model of an urban environment. Users interact with physical building models in order to create and alter this computational model of an urban environment. The physical properties of a building model suggest that a user can grab it and then place or remove it from the table. Also, the shape of a building model suggests to users how to place the building upon the surface (roof at the top, floor at the bottom). We also consider the distance-tool as a token in the URP system because, it represents the computational function 'displayDistance'.

Alternatively, in some cases we may consider different body parts as tokens. For example, in a TUI where each finger represents a different tune, we consider each finger as a token.

TUIML depicts tokens using a diagrammatic notation that convey their shape and orientation but abstracts away physical properties such as texture, and size. Such properties could be specified using a secondary textual notation. Tokens are depicted as simple geometrical shapes that contain a variable. The shapes used to represent a token vary: TUIML offers abstract geometrical shapes such as rectangles or circles for representing tokens, however, a TUI developer may choose to use shapes that are more complex and resemble the actual look of a certain object. Each shape that is used to represent a token should contain a graphical symbol that represents the variable binded to this token. Variables of type digital information are represented using a small circle while variables of type computational function are represented using a small rectangle. Figure 1 shows TUIML representations of some of the tokens used at URP.

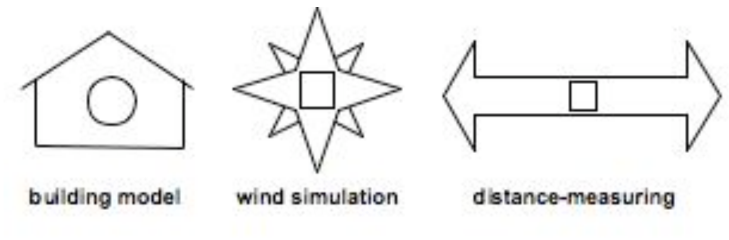


Figure 1, TUIML representations of URP tokens.

TUIML also depicts active tokens. Figure 2, shows TUIML representations of blocks which serve as active tokens augmented with attraction, repulsion and vibration.

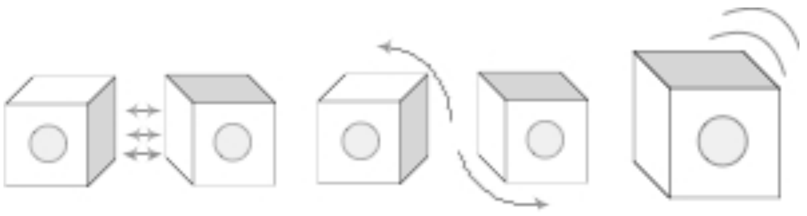


Figure 2, active tokens (blocks) augmented with attraction, repulsion and vibration.

When a TUI contains several instances of the same token type (e.g. An Urp system that contains eight building models), TUIML enables TUI developers to mark the number of possible token instances (rather than drawing each of them separately). Figure 3 shows how to mark the number of possible token instances.

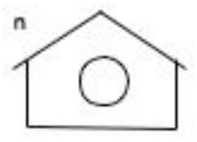


Figure 3, a building model token that could possibly have *n* instances.

A Constraint is a physical object **that limits the behavior of a token** with which it is associated. The physical properties of the constraint **guide the user in understanding how to manipulate the token and how to interpret configurations of token and constraints**. A constraint limits a token's behavior in one or more of the following three ways:

1) Affording to the user how to manipulate (and how not to manipulate) an associated.
2) Confining the physical interaction range of associated tokens.
3) Serving as a reference frame for the interpretation of token and constraint compositions.

For example in the Urp[] system we consider the surface as a constraint because it confines the interaction with building models to take place within its dimensions. The surface also serves as a reference frame for interpreting the position of buildings models and the space between building models. Alternatively, different body parts can be viewed as constraints because they constrain movement for example, in terms of reach, and rotations.

It is important to note that **a certain physical object may serve as a token, a constraint or both.** For example, we consider a building model as a token. However, in the context of measuring distance between two building models, a building model can be considered as a constraint because it limits the range of the distance measuring interaction.

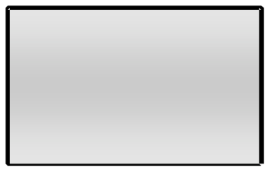
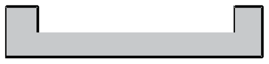
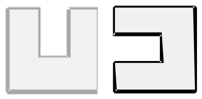
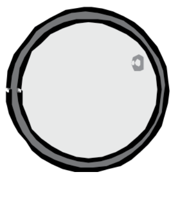
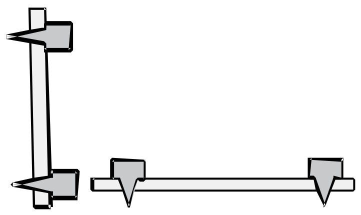
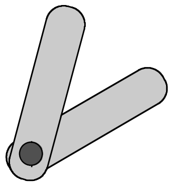
In order to capture the physical relations between tokens and constraints and hint toward how a token could be manipulated in respect to a constraint, TUIML depicts constraints using a diagrammatic notation that convey their shape, orientation and size relative to tokens. Again, the TUIML notation abstracts away physical properties such as texture, color and absolute size, (these can be specified using a secondary textual notation). Table 1 shows TUIML representations of common and widely used constraint types. For each constraint type we present its TUIML representation and list the physical relations possible between this constraint and associated tokens. It is important to note that some constraints may be associated with several tokens. For example, a surface could contain several building models, the manipulation of each is constrained by the surface but also by the presence of the other building models. Thus, for such constraints table 1 lists the relations possible between an associated token and the constraint (e.g. presence) as well as between all associated tokens (e.g. order).

TUI developers can easily extend TUIML notation by adding constraint types to table 1 and list the physical relations they support.

A TAC (Token And Constraints) is the **relationship between a token and one or more constraints**. Often, this relationship is temporary. TAC relationships are defined by the TUI developer and are created when a token is physically associated with a constraint. For example, in the URP system, we consider the combination of a building model upon a surface as a TAC. Such a TAC is created whenever a building model (i.e. token) is added to the surface (i.e. constraint). If five buildings are located upon the surface, we say that there are five instances of a building model-surface TAC.

Interacting with a TAC involves physically manipulating a token (in a discrete or a continuous manner) in respect to its constraints. Such interaction has computational interpretation. Thus, the manipulation of a token in respect to its constraints results in modifying both the physical and digital states of the system. Manipulation of a token outside its constraints has no computational interpretation. Only when a token is associated with constraints does its manipulation have computational interpretation. Thus, we can view **TAC objects as similar to Widgets** because they encapsulate both the set of meaningful manipulations users can perform upon a physical object (i.e. methods) and the physical relations between tokens and constraints (i.e. state). For example a TAC that consists of a building model and a surface encapsulates the following manipulations: addBuilding, removeBuilding and moveBuilding as well as the following physical relations: position, orientation, proximityto(Building b), leftof(Building b) etc.

TAC relationships may have a recursive structure so that a given TAC can serve as a token or a constraint for other TACs. For example, when two building models are connected using a distance-measuring tool we can view this configuration as a nested TAC, where the first TAC which consists of a distance-measuring tool (i.e. token) connected to two building models (i.e. constraints) then serves as a token which is constrained by a surface.

Constraint	**Notation**	**Physical relations**
Surface		Identity, presence, position (x,y,z), orientation, proximity, spatial relations, order, number, group, containment.
Rack		Identity, presence, order, left of, right of, proximity.
Indentation		Identity, presence, orientation.
Knob		Identity, position, orientation.
Slider		Identity, presence, position.
Joint		Identity, position.

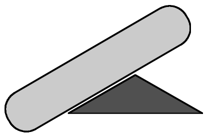
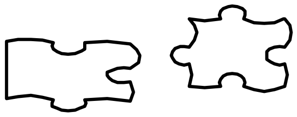
Pedal		Identity, position.
Connector		Identity, connection, left of, right of, above, below.

Table 1, TUIML representations of commonly used constraints.

Depicting TAC relationships is about capturing configurations, connections, spatial layout, shape and orientations of tokens and constraints elements. To accomplish this, we **combine TUIML elements of tokens and constraints into TACs.** For example, figure 4 depicts two TAC relations within the URP interface. The first, consists of a building model and a surface. The building model is marked with an m above its left corner to mark that m building models can be located upon this surface. The second TAC, is a nested TAC, which consists of a pair of building models connected by a distance-tool upon a surface. The dotted line represents that we see the pair of buildings and the distance-tool as a separate TAC type that could have n simultaneous instances. Meaning, assuming there is a sufficient number of building models and distance-tools, n distance measurements can be performed simultaneously.

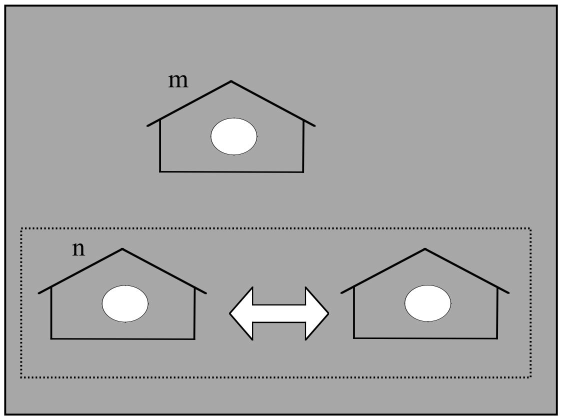


Figure 4, TAC relations in the URP interface.

The set of constructs presented in this section is sufficient to describe the functionality and structure of a broad range of TUIs . In the TAC paradigm paper **you can find examples** of several TUIs described in terms of tokens and constraints.

Having presented a set of constructs, we now explain how to use them to describe the structure of a TUI.

1.2 Describing the structure of a TUI using TUIML

To specify the structure of a TUI using TUIML, a TUI developer first **defines a set of tokens**. For each token, she provides a visual representation and a list of properties. Then, the TUI developer creates a TAC palette, a table that contains all possible TAC relationships of a certain TUI.

The listing of possible TAC relationships is done in terms of *representation, association* and *manipulation*. *Representation* refers to the binding of a token to an application variable and the selection of constraints. Association refers to the physical association of a token and a set of constraints. Finally, *Manipulation,* refers to the actions a user may perform upon a TAC.

TAC relationships are defined by the developer, but may be instantiated by either the user or the developer. Typically, instantiation of a TAC is initiated in response to a discrete event. For each TAC which may be instantiated or destroyed at run time, the TUI developer must define the discrete actions *add* and *remove*, which instantiate or destroy the TAC. These actions may also have additional computational effect on the TUI beyond simply instantiating and destroying TACs. Figure 5 shows the specification of a Building Model token. Figure 6 depicts a partial TAC palette for URP.

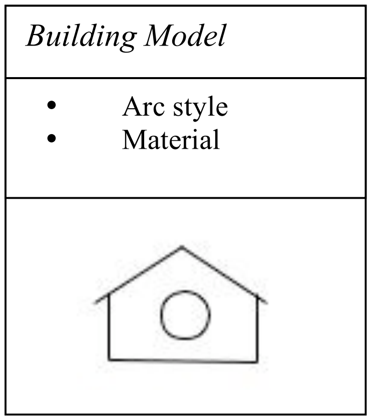


Figure 5, specification of a Building Model token.

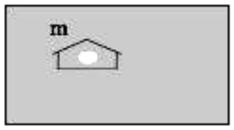
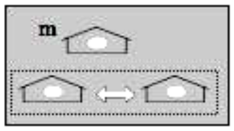
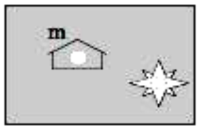
TAC	Representation			Association	Manipulation	
	Variable	Token	Constraint	TAC graphics	Action	Response
1	building	building model	surface other buildings		Add	displays shadow according to time.
					Remove	Removes related info from display
					move	Updates display
2	distance	Distance tool	Two buildings Surface		Add	Displays distance
					remove	Hides distance
3	wind simulation	Wind tool	Buildings surface		Add	Displays wind
					Remove	Hides wind
					move	Updates wind

Figure 6, a TAC palette for URP.

Having introduced a set of constructs and a mechanism for describing the structure and functionality of TUIs, the next section will explain how to model the underlying behavior of a TUI using TUIML.

2. Describing Behaviors Using TUIML

Current user interface specification languages mainly rely on event-driven models for specifying and programming the current generation of graphical user interfaces. However, they seem as a wrong model for explicitly specifying continuous and parallel behaviors, which are common in TUIs. TUIML is intended to provide TUI developers with means for describing the behaviors of a TUI at a high-level, close to the way users view such interaction. Thus, as TUIs convey a sense of continuous and parallel interaction to users (for example by allowing multiple users to simultaneously move multiple objects upon a surface), TUIML is concerned with capturing these interaction qualities *directly*.

2.1 An Interaction Model for TUIs

In order to develop a specification language capable of describing the underlying behavior of TUIs at a high-level, it is first necessary to identify an interaction model that directly addresses continuous and parallel interaction.

TUIML recognizes two fundamental event types that repeat throughout an interaction with a TUI and may cause a *mode change* in an interface: 1) *dynamic binding* of digital information to physical interaction objects (i.e. when users couple information to physical objects of their choice at run-time.) 2) *physical association* of objects (e.g. when users physically add/connect physical interaction objects to each other). Both event types cause a mode change in an interface because they either alter the meaning of an interaction or modify the range of possible interactions. For example, consider the URP system, when a second building is added to the surface, the set of possible interaction is modified – users can not only add, remove and move a building model but also measure the distance between two buildings. Thus, TUIML identifies the basic structure of a tangible dialogue as **a** *sequence of modes or high-level states***.** However, within each high-level state, multiple users may interact with the system, in a discrete or continuous manner, in parallel or consecutively. Hence, TUIML views the underlying behavior of a TUI as *a collection of orthogonal user interactions, each represents a thread of functionality and is nested within a high-level state*.

This leads to a two-tier model for describing the behavior of TUIs. Our two-tier model contains a *dialogue tier* and *an interaction tier*:
The *dialogue tier* provides an overview of the tangible dialogue structure. It consists of a set of high-level states and transitions. A high-level state represents a context in which a set of meaningful user interactions may occur (in parallel or consecutively). A transition represents an event that changes the context in which interactions take place thus leads to a state change.

The *interaction tier,* consists of a collection of orthogonal user interactions. It provides a detailed view of each user interaction that represents a thread of functionality. For each such interaction, it describes: its decomposition to parallel and continuous manipulations, the interaction objects it employs as well as its affect on the digital and physical states of the system. These two tiers, the dialogue tier and the task tier, communicate via a shared memory. We represent the dialogue tier using a *dialogue diagram* and the task tier using *a collection of interaction diagrams* nested within the dialogue diagram. Figure 7 describe our two-tier model.

Following we further discuss the key concepts of this model and introduce a notation for each concept.

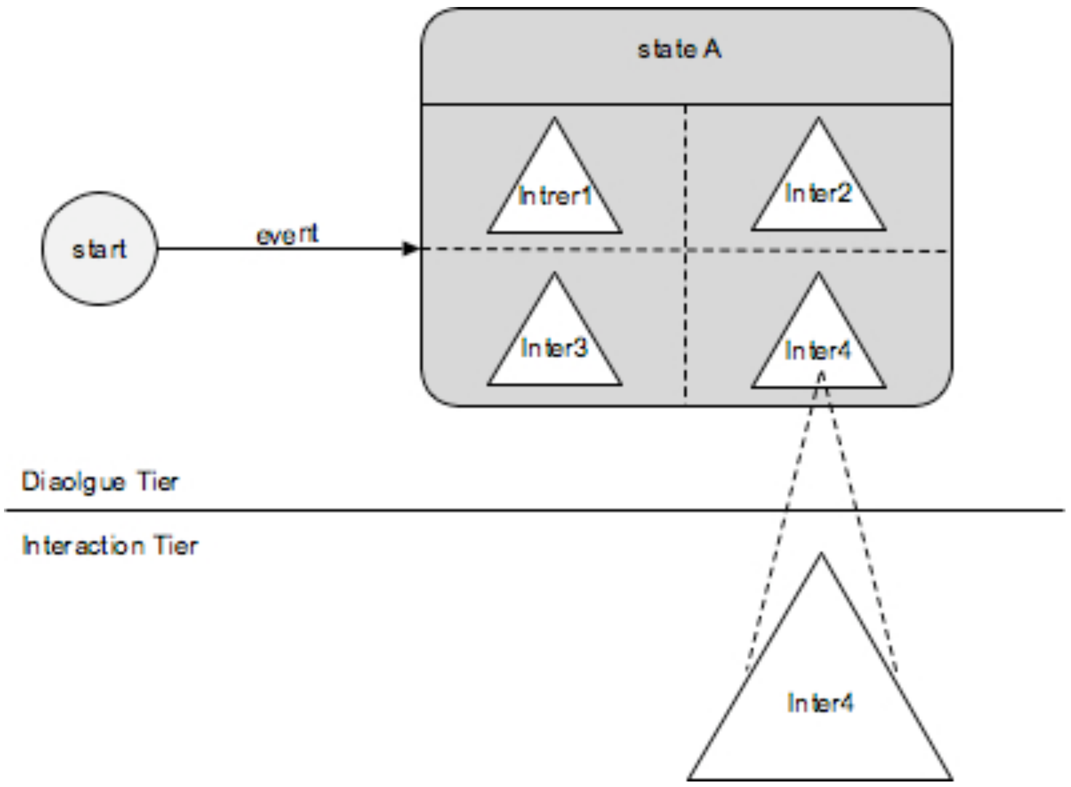


Figure 7, A two-tier model for describing the behavior of TUIs. The dialogue tier (at the top) describes a set of high-level states a TUI can be in. Each high-level state contains a set of interactions that can be performed when the system is in this state. A 'zoom in' into the dialogue tier reveals the interaction tier: a collection of individual interaction diagrams nested within each state. Each of these diagrams is represented by a triangle. The internal structure of an interaction diagram will be explained later in this section.

2.2 The dialogue-tier

The *dialogue-tier* provides a high-level overview of a tangible dialogue structure. It consists of a set of high-level states and a set of transitions. The dialogue diagram draws upon the Statechart notation, a state-based notation that enables to express concurrency within a state.

High-level States

Rather than describing the state of a TUI at every turn, what would make the dialogue specification excessively complex and would lead to a state explosion, a high-level state captures a context in which a certain set of user interactions are meaningful. These user interactions are orthogonal and could be performed in parallel assuming their pre-conditions are satisfied.

For example, we identify three high-level states in the URP system: no building models located upon the surface, one building model located upon the surface and at least two buildings located upon the surface. When no building models are located upon the surface there are no meaningful interactions that users can perform beside adding a building to the surface what cause to the system to move to a new state: a single building is located upon the surface. In this state users can move the building model, change its orientation, perform a wind simulation, change the building model material and update the digital shadow of the building by changing the time of the day in the urban model. When two or more buildings are located upon the surface, users can perform all the interactions listed above as well as measure and change the distance between two buildings.

Formally, a high level state encapsulates three elements: an internal state, a physical state and a set of meaningful interactions. Table 2 summarizes the elements included in a high-level state.

Element	**Description**
Internal state	A vector of the current values of application variables.
Physical state	A vector of instantiated TAC relationships.
Interactions	A set of interactions that could be performed in parallel or consecutively within this state.

Table 2

In the TUIML dialogue-diagram, a high-level state is denoted by a rounded rectangle symbol and each of the meaningful interactions contained in a state is denoted by a triangle symbol separated from others by a dashed line. Figure 8 shows a TUIML representation of a high level state. When informally modeling a dialogue tier it is sufficient to visually represent a high-level state and the set of meaningful interactions it contains. However, for a more detailed specification a table that describes the internal and physical states should accompany the visual description.

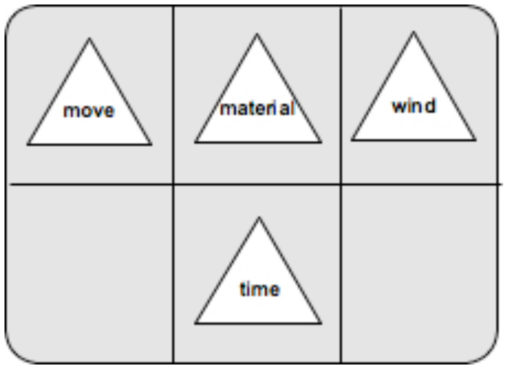


Figure 8, a visual representation of an URP high-level state.

Similar to a Statechart, a dialogue diagram may have initial and final states. An initial state is the one that the TUI is in when it is first activated. A final state is one in which no transitions lead out of.

Transitions

A transition between high-level states occurs as a result of a discrete event that changes the context in which interactions with a TUI take place. A transition is associated with an event and may also be associated with a condition and a response. In order for a transition to 'fire', a discrete event should occur and its associated condition must be true. In addition to causing a high-level state change, an event that is associated with a transition may cause a response such invocation of application functions combined with digital and physical output.

A response is a result of the transition event. For example, when a building is added to the URP surface the system may move to a new state (depends how many building are already located upon the URP surface) and a digital shadow for the new building model is projected upon the surface.

In a TUI, several sources may produce transition events. Thus, TUIML introduces four transition types, based on their source. TUIML represents a transition as a labeled arrow. Table 3 introduces the graphical representation for each transition type. Figure 9 shows the format for transition labels.

Source	**Description**	**Representation**
Timer	Generated by a timer.	———●
System	Generated by the system.	———◆
User Interaction	Generated intentionally by a user.	———▶
Implicit Interaction	Generated implicitly by a user.	———>

Table 3, representations of different transition types.

event C: condition / R: response

Figure 9, format for transition labels.

Having established a notation for a dialogue diagram, figure 10 shows the dialogue diagram for the URP system. The diagram consists of three high-level states: an initial state where no building models are located upon the surface, a state where one building model is located upon the surface and a state where at least two building models are located upon the surface. Each of these high-level states contains a set of interactions that users can complete (in sequence or in parallel) while the system is within this state. In the URP system the transitions from one high-level state to another (or back to the same state) are all a result of user interaction.

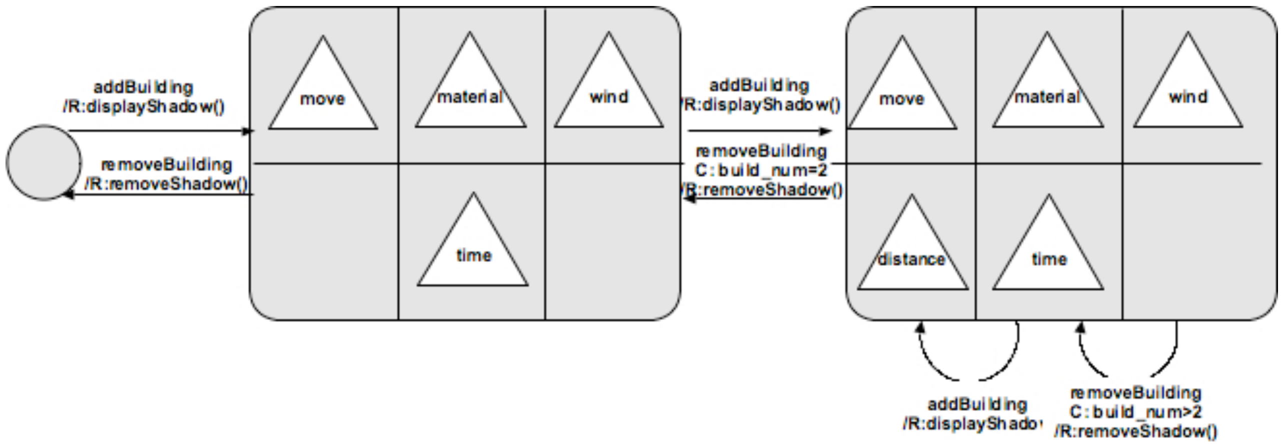

addBuilding /R:displayShadow()
removeBuilding /R:removeShadow()
move
material
wind
time
addBuilding /R:displayShadow()
removeBuilding C:build_num=2 /R:removeShadow()
move
material
wind
distance
time
addBuilding /R:displayShadow
removeBuilding C:build_num>2 /R:removeShadow()

Figure 10, a dialogue diagram for URP.

2.3 The Interaction-Tier

While the dialogue-tier provides an overview of the tangible dialogue structure, the interaction-tier provides a detailed view of each user interaction that represents a particular thread of functionality.

A tangible interaction typically consists of a set of discrete actions and continuous manipulations performed consecutively or in parallel upon physical interaction objects. For example, the interaction aimed at distance measuring in the URP interface consists of a sequence of two discrete actions: connecting two buildings using a distance-tool (results in displaying the distance between the two buildings) and disconnecting the buildings when the distance display is no longer required. Alternatively, to drive a car down the road users perform two continuous manipulations in parallel: controlling the steering wheel and adjusting the gas pedal. Some tasks involve both discrete actions and continuous manipulations. For example, when a user shoots an enemy while flying an airplane in a video game.

The interaction-tier depicts the decomposition of tangible interactions into a set of discrete actions and continuous manipulations and specifies the temporal relations between them. For each action or manipulation (discrete or continuous) the diagram describes the pre-conditions needed to be satisfied in order for it to take place, and the post-conditions it causes in terms of its affect on the digital and physical states of the system.

It is important to note that during different phases of the design process it is helpful to model an interaction using different granularity levels. For example, early in the design process the task of distance measuring may be modeled using low granularity so that the first action users perform in order to display the distance between two buildings is connecting them using the distance tool. Alternatively, later in the design process when implementation constraints are known, applying higher granularity will divide the action of connecting two buildings into two discrete actions: touching the first building using the distance tool and then touching the second building.

TUIML represents the interaction-tier as *a collection of interaction diagrams each nested within a high-level state*. It depicts interaction diagrams using a graphical notation that directly captures parallel and continuous interaction as well as express both physical and digital states of the system. The TUIML notation is inspired from Petri nets because Petri nets provides an appealing graphical representation for specifying systems that exhibit parallel activities.

The basic structure of a TUIML interaction diagram is similar to a Petri net. It comprises two types of nodes*: places* and *transitions* that are connected by flow relations (i.e. direct arcs). Places represent conditions in terms of physical or digital configurations and transitions represent discrete events. However, we modified and enriched the PN notation in several ways. Among these changes:

1. Places are represented using a special notation capable of expressing relationships between physical objects.
2. An additional node type is introduced in order to represent continuous

manipulations.

3. The diagram is divided into two areas that represent the physical and the digital worlds correspondingly.

Following we describe the structure of an interaction diagram in further detail. In order to clearly introduce the different elements of the interaction diagram we will use the URP wind-simulation interaction as a leading example. Figure 11 shows the wind-simulation interaction diagram.

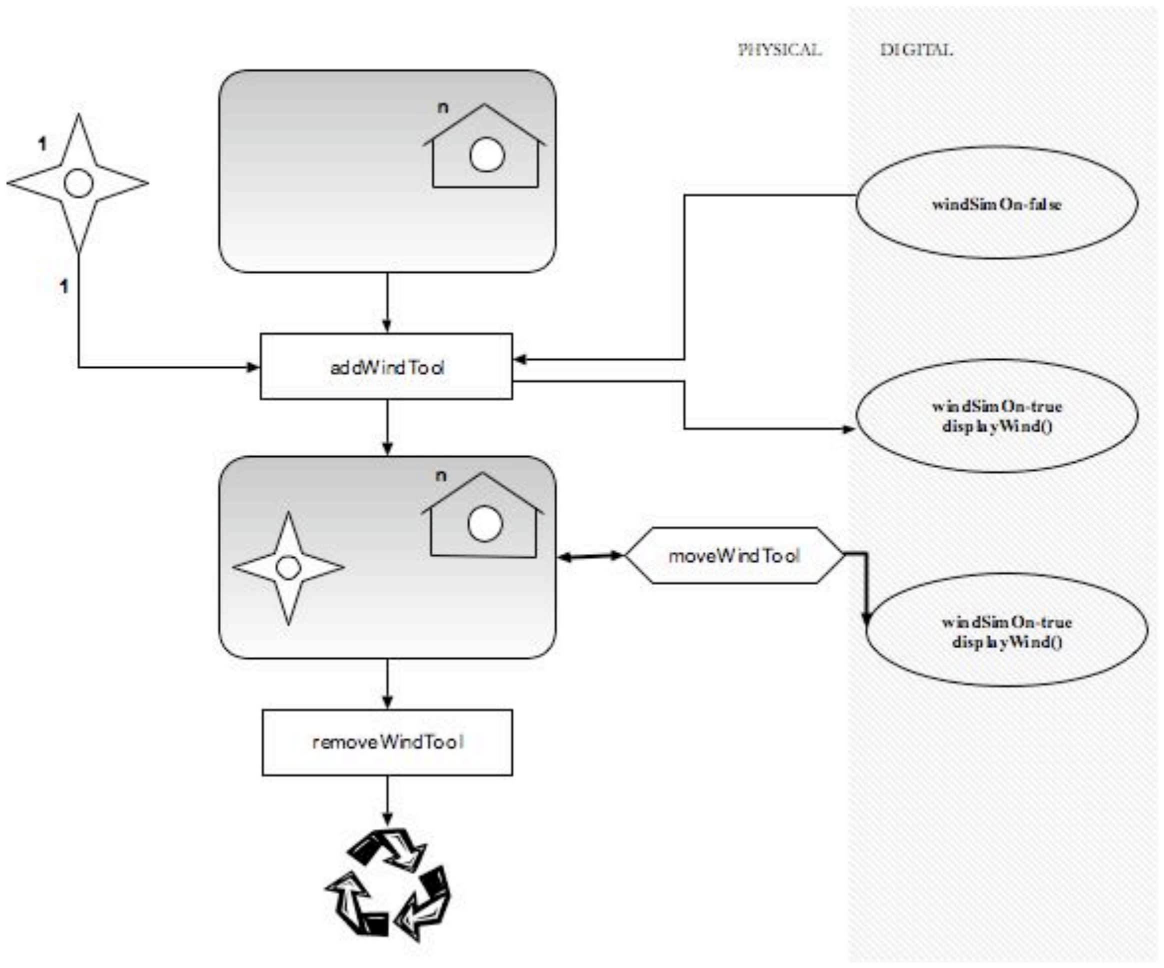


Figure 11, an interaction diagram for URP's wind simulation.

Places, transitions and flow relations

In the modeling of a tangible interaction, *transitions* represent discrete events. For an event to occur, it may be necessary for certain pre-conditions to hold. The occurrence of an event may cause its pre-conditions to cease to hold and may cause post-conditions to become true. In an interaction diagram, a transition is represented using a box, the inputs of a transition are the preconditions of the corresponding event; the outputs are the post-conditions. The occurrence of an event corresponds to the 'firing' of the corresponding transition. The wind-simulation interaction diagram (see figure 11) contains two transitions: addWindTool and removeWindTool.

Pre and post conditions are represented by *places.* In a TUI, pre and post conditions may relate to both the physical state of the system and the digital state of the system. For example consider the transition addWindTool in the wind-simulation interaction diagram (see figure X). In order for this transition to result in the activation of the wind simulation the following pre-conditions must be true: 1) the URP surface should contain at least one building 2) The URP system should contain a wind-tool, but it shouldn't be located upon the surface 3) the digital flag 'windSimulationOn' should be set to false. This transition also causes the following post-conditions: a physical wind interaction object is located upon the surface, the digital flag 'windSimulationOn' is set to true, and a graphical visualization of a wind flow is displayed upon the surface.

To express pre and post conditions that relate to both physical and digital configurations, TUIML represents places using a mixed notation. Conditions that relate to the physical state of the TUI are expressed in terms of tokens, constraints and TAC objects and denoted using their graphical representations. While conditions that relate to the digital state of the system are represented using text enclosed within an oval node. In the example of the wind-simulation interaction (see figure 11), the physical pre-conditions for the addWindtool transition are modeled using a TAC object that represents building models upon a surface, and a token object that represents a wind-tool. The digital pre-condition are specified within an oval node. Similarly, the physical post-condition is modeled using a TAC object that contains building models as well as wind-tool upon a surface while the digital post-conditions are textually specified within an oval node. Note that physical conditions are depicted within the physical-world area while digital conditions are depicted within the digital-world area.

Places that represent physical conditions could be given a *marking*. A marking is a number that is specified in the upper left corner of a token or TAC objects and signifies how many instances of that object may exist in the TUI system. Consider the TAC object that serves as pre-condition for the addWindTool transition (see figure 11). It consists of a building model token upon a surface. Its building model token has a marking of n, expressing that n building models could be located upon this surface assuming n is bigger than one and smaller than ten. The wind-tool, which also serve as a pre-condition for the addWindTool transition, has a marking of 1. The overall marking of an interaction diagram reflects its state. If not specified otherwise, the default marking of places is one.

Transitions are connected to places using *Flow relations*, which are represented by directed arcs. When a flow relation connects a place that represents a physical condition to a transition, it may be associated with a number (i.e. weight function). This weight function expresses how many instances of a certain physical interaction object are required in order to fire the associated transition. For example, in order to fire the transition addWindTool (see figure 11) exactly one wind-tool is required. If no weight is specified, a default weigh value of one is applied.

During the simulation (execution) of an interaction diagram, its initial marking changes according to the sequence of fired transitions, to reflect changes in the diagram state. For example, following the firing of the transition addWindTool, the wind-tool is located upon the surface. Thus, the initial marking of the interaction diagram that reflects the availability of one 'free' wind-tool changes to reflect that in the current state of the system there is no available wind-tool. Naturally, it implies that no two wind simulations could be performed in parallel.

Finally, transitions that conclude the sequence of firing transitions, are connected to a special output place called ‘recycler’ that has no outgoing flow relations. Following the execution of such a transition, the initial marking of the interaction diagram is recovered, and the initial pre-conditions are set to true. Meaning all physical objects that were employed during the execution of this interaction are released. For example, the transition removeWindTool is connected to a recycler state. Following its execution the wind-tool is removed from the surface and the flag windSimulationOn is set to false. Thus, users can perform the wind simulation interaction once again.

Having described the basic structure of an interaction diagram, we can now explain how we integrate continuous interactions into the interaction diagram structure.

Manipulations

Continuous manipulations give users continuous feedback in response to continuous input. Hence, they are not appropriately modeled by transitions, which represent discrete events. In order to model continuous interaction explicitly within the TUIML interaction diagram, we introduced additional node type to the diagram called *a manipulation* that is depicted using an hexagon.

Similar to a discrete event, a manipulation may have pre-conditions related to both the physical and digital states of the system. The continuous manipulation may last while these pre-conditions hold. A manipulation may also produce *continuous* digital output and change in the digital state of the system. Thus, it may be connected, using directed arcs, to places that represent digital configurations. While a manipulation may produce physical output, it cannot change the physical state of the system in terms of tokens and constraints configurations, because such configurations are created or destroyed when physical objects are added to or removed from other physical objects (i.e, in response to discrete events). Rather, a continuous manipulation occurs *within* a particular configuration of tokens and constraints. Thus, places that represent physical conditions serve as both pre and post-conditions of an associated manipulation and are connected to a manipulation node using a bi-directional arc. To emphasize the continuous link between a manipulation node, its pre and post conditions, TUIML uses specialized thicker arcs to represent bidirectional, ingoing or outgoing continuous flow relations.

In the wind-manipulation interaction diagram (see figure 11), there is one manipulation node, that refers to the continuous manipulation moveWindTool. Following the execution of the addWindTool transition, a wind tool is located upon the surface. Users can then continuously move the wind tool upon the surface in order to change the direction of the wind. A pre-condition for this manipulation is the physical configuration described as a TAC, which contains building models and a wind-tool upon a surface. The movement of the wind-tool upon the surface does not change this physical configuration (it only changes the position of the wind-tool in respect of the surface), thus, a bi-directional arc connects the moveWindTool manipulation to the place, which represents this physical configuration. The moveWindTool manipulation causes a continuous update of the digital wind simulation display. This is specified within a place that is connected to the moveWindTool manipulation using an outgoing arc.

A manipulation may also fire a transition in response to a certain variable crossing a threshold. For example when a users slides an object away from another object, the corresponding manipulation fires an event as a result of the distance between the objects crossing a certain threshold. In such cases, the manipulation node is connected using an outgoing flow relation (i.e. an outgoing arc) to a transition that represents the threshold-crossing event.

2.4 Summary

The two-tier model and diagrams presented here are aimed to enable TUI developers to describe and discuss a TUI behavior from a point of view closer to the users' rather than to the exigencies of implementation. The dialogue diagram is intended to bring out the big picture, and assist TUI developers to assure that functionality is complete and correct prior to implementing a fully functional prototype. An interaction diagram allows TUI developers to focus on a specific thread of functionality. It provide users with means for addressing design concerns related to parallel interaction such as multiple access points and visibility, understanding the consequences of users' input on both the digital and physical states of the system as well as considering issues related to combined discrete and continuous inputs. The comparison of interaction diagrams allows TUI developers to consider alternative designs and create consistency of interaction syntax within an application.

3. An Example: The Marble Answering Machine

One of the earliest illustrations of interlinking the physical and digital worlds is provided in the design of the Marble Answering Machine (MAM) []. It was designed and prototyped by Durrell Bishop, while a student at the Royal College of Art, in order to explore ways in which computing can be taken off the desk and integrated into every day objects. However, it was never fully implemented. We have selected the Marble Answering Machine as an example, because it is a simple and elegant example of a TUI.

In the Marble Answering Machine, marbles represents incoming voice messages. To play a message, a user grabs a message (marble) and places it in an indentation on the machine. To return a call, the user places the marble within an indentation in an augmented telephone. To store a message, the user places a marble in a dedicated storage saucer – different users may have different saucers.

Figure 12, presents the TAC palette for the MAM. Visually specifying the MAM structure highlights the use of physical constraints to enforce physical syntax (see X). For example, TAC 2, consists of a marble and a replay indentation. The shape of the replay indentation affords the placement of a single marble within its dimensions. Meaning, only a single message could be played at a certain time. The visual specification of a TAC could also assist in comparing alternative designs. For example, within the scope of the MAM, we can compare the structure of TAC 1, a marble within a message queue, with the structure of TAC 4, a marble within a storage saucer. While the physical properties of a rack (used for representing the message queue) imply the following relations: presence, order and number, the physical properties of a storage saucer only imply the relations of presence and number. As such, a user approaching his storage saucer is not aware to the order in which his incoming messages have arrived. Replacing the storage saucer with other constraint types provided by TUIML, such as a rack or a series of indentations allows the TUI developer to consider alternative designs.

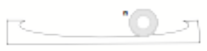
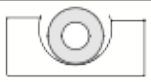
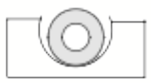
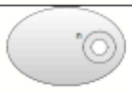
TAC	Representation			Association	Manipulation	
	Variable	Token	Constraint	TAC graphics	Action	Response
1	Message	Marble	MessageQueue		Add	
					Remove	
2	Message	Marble	ReplayIndentation		Add	Play Message
					remove	Stop Playing
3	Message	Marble	CallbackIndentation		Add	Dial back
					Remove	Disconnect call
4	Message	Marble	StorageSaucer		Add	
					Remove	

Figure 12, MAM TAC palette.

Figure 13, presents the dialogue diagram of the MAM interface. This diagram depicts two different transitions types: those generated by system events (e.g. incoming calls) and those generated by users (e.g. remove marble).

Comparing the *play* and *call back* interaction diagrams (figure 14 and 15), highlights a consistent syntax across these two interactions. Also, considering the interaction objects required for completing each of those interactions (a marble and a play indentation for playing, a message and a marble and a call back indentation for calling back) shows that these two interactions could take place in parallel assuming the MAM contains at least two different messages.

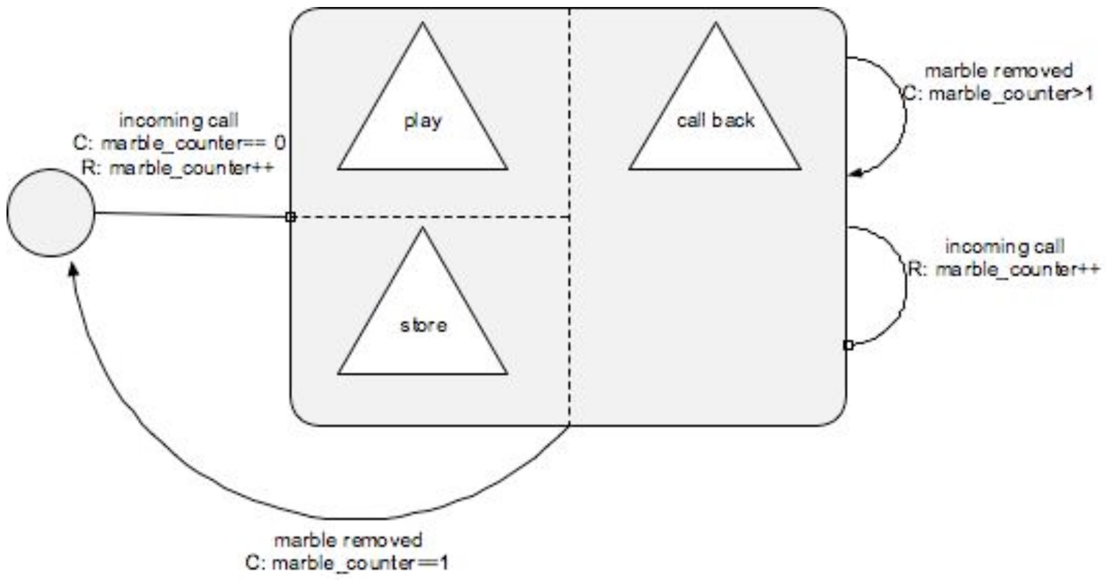
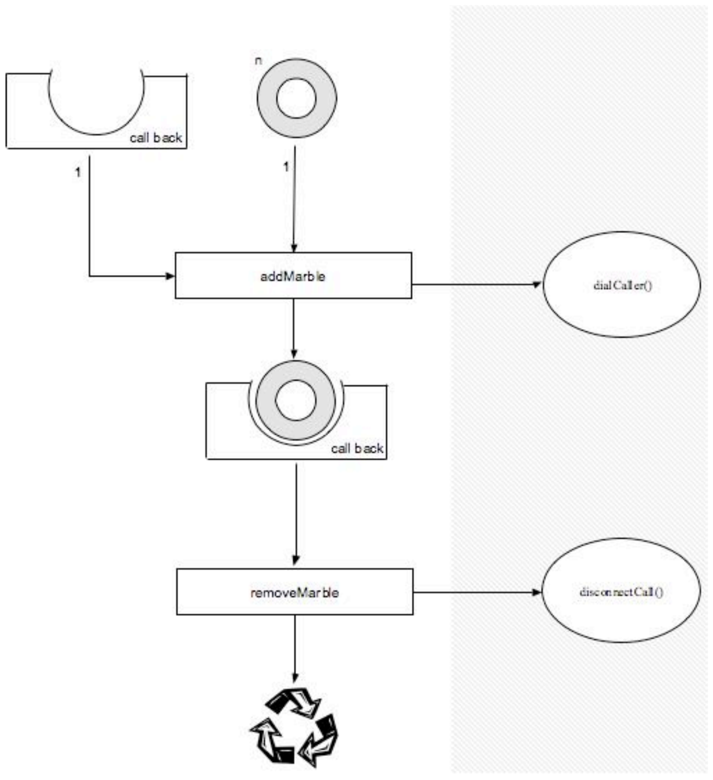


Figure 13, MAM dialogue diagram.



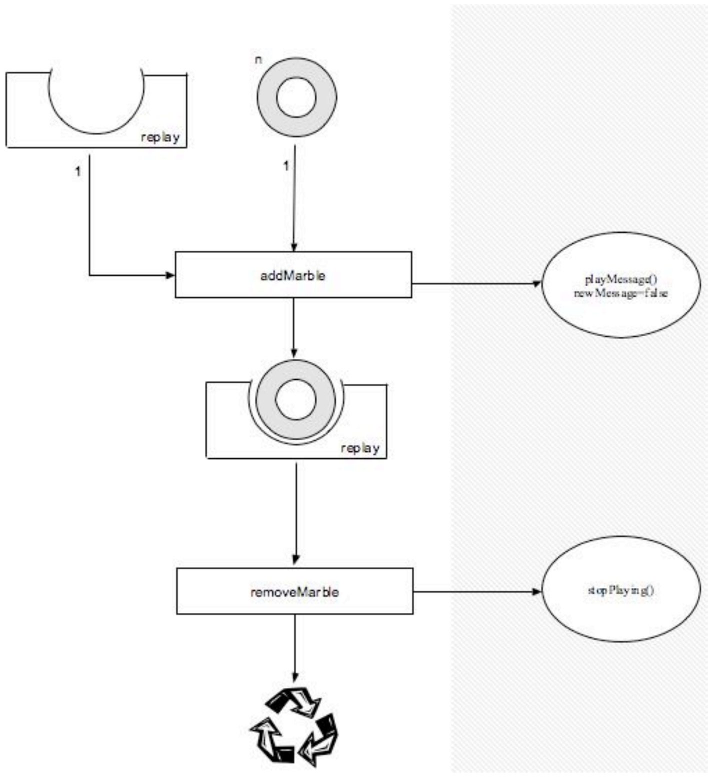


Figure 14, call back interaction diagram. **Figure 15,** play interaction diagram.