GENERIC STRATEGIES FOR MANIPULATING GRAPHICAL INTERACTION OBJECTS: AUGMENTING, EXPANDING AND INTEGRATING COMPONENTS

D. Akoumianakis, G. Milolidakis, D. Kotsalis and G. Vellis
Department of Applied Informatics & Multimedia, Faculty of Applied Technologies
Technological Education Institution of Crete, P.O. Box 1939 Iraklio, Crete, GR 710 04, Greece

Keywords: User interface toolkits, augmentation, expansion, integration, abstraction.

Abstract: This paper presents the notion of (user interface development) platform administration and argues for its increasing importance in the context of modern interactive applications. Platform administration entails strategies for manipulating diverse interaction components. Four such strategies are elaborated – namely augmentation, expansion, integration and abstraction – which collectively constitute the ingredients of a platform administration process. The paper describes both the rationale for these strategies in the context of user interface development and their implementation details, as currently realized in an ongoing R&D project.

1 INTRODUCTION

Over the past two decades, the development of graphical user interface development toolkits has been continuous, addressing a variety of aspects including cutting-edge issues in 2D graphical interaction e.g., Piccolo (Bederson et al., 2004) and its predecessor Jazz (Bederson 2000), information visualization e.g., prefuse (Heer et al., 2005), etc. Although toolkits are popular and widely used, they pose several constraints to user interface developers related to the type, range and scope of implemented widgets.

An alternative user interface development method makes use of abstract notations and mark-up languages – typically dialects of XML – to facilitate mapping of abstract components to platform-specific toolkit libraries by delegating the display to a platform-specific renderer (Lee et al., 2006).

Each approach has relative merits and drawbacks, while they may also conflict at times. Some of the advantages of toolkit programming include the capabilities to build improved interaction techniques and to construct novel widgets and interaction object hierarchies. The disadvantage is that realizing such capabilities is demanding and programming-intensive task. On the other hand, approaches based on device-independent markup languages are increasingly supported by tools, they are less demanding in terms of programming skills, while they adopt some sort of abstraction-based mechanism to make a step towards 'write once, run everywhere' user interfaces (Perry et al., 2001). As for disadvantages, they are still in an infant state, while their multi-platform capability typically does not easily account for the improvements introduced by toolkit programming-based techniques.

Irrespective of the development approach, one problem which is frequently faced by designers and developers of interactive systems is that specialized applications often require widgets that are unique to a particular problem. Such domain-specific or legacy widgets are typically not directly supported by popular toolkits. In some cases, they can be created from the simpler native building blocks depending on the extensibility features offered by a specific toolkit. Nevertheless, the creation of such custom widgets is far from trivial and frequently assumes ad hoc practices.

In this paper, we aim to describe the core elements of a user interface development process intended to cope with challenges such as the above in a systematic manner. We will present key constituents of this process as well as how they have been instantiated using Java / Swing in the course of recent developments. To this effect, examples of

Akoumianakis D., Milolidakis G., Kotsalis D. and Vellis G. (2008).
GENERIC STRATEGIES FOR MANIPULATING GRAPHICAL INTERACTION OBJECTS: AUGMENTING, EXPANDING AND INTEGRATING COMPONENTS.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - HCI*, pages 21-29
DOI: 10.5220/0001679700210029

new primitive and composite widgets are presented, together with their implementation details.

The reminder of the paper is structured as follows. The next section motivates the problem at hand and relates it to on-going research and development activities. Then, the platform administration process is overviewed in terms of constituent activities, their rationale and intended scope. This is facilitated by illustrative examples of running prototypes and brief presentation of their technical features with reference to Java's Swing. In the last section, we summarize the contributions of this work, relate them to other similar efforts in the relevant literature and draw some conclusions and directions for future work.

2 PROBLEM DESCRIPTION

All user interface development toolkits offer a limited number of widgets. For certain applications the supported widget set may not suffice to provide the interactive embodiment demanded by designers. As a partial solution to the problem, toolkits offer a set of custom widgets and / or mechanisms for building new custom widgets. However, there may be problems and applications which cannot be adequately served by custom widget construction techniques. In such cases, developers may consider the development of a new dedicated toolkit implementing alternative spatial semantics and / or the integration of a third-party library which offers alternative or more appropriate interaction components. In both these cases, the pressing issue is on the interoperability between the base toolkit and the third-part library or the new toolkit. These considerations pose new challenges for user interface developers which increasingly need to be prepared to manage diverse collections of interaction resources.

Our interest in these issues dates back to early accounts of universally accessible interactions (Stephanidis et al., 1997; Stephanidis et al., 2001) and the development of multiple metaphor environments (Akoumianakis et al., 2003). Recent research and development activities have renewed and extended this interest, resurfacing some of the limitations of widely available and cutting edge 2D graphical user interface development toolkits (Akoumianakis et al., 2008, Akoumianakis, 2008). Consider for example the case of synchronizing user interfaces across multiple-devices so as to allow collaborative exploration of large volumes of community data to identify common patterns or to assess behavioral relationships between the data (e.g., conditional aggregation-desegregation patterns). Conventional 2D graphical toolkits do not offer the required support to build such user interfaces effectively and efficiently. Thus, developers either sacrifice usability or adopt ad-hoc and one-off solutions.

Currently, we are facing such problems in the context of an R&D project, namely eKoNEΣ (see acknowledgement), aiming to construct and test a pilot application of an electronic village of local interest on tourism (Akoumianakis et al., 2007, Akoumianakis et al., 2008). Inhabitants and visitors of the electronic village form dynamic squads (on-line communities of practice) engaging in a variety of social interactions (i.e., establishing and maintaining sense of community, negotiating goals, resolving conflicts, establishing norms) so as to develop new added-value products and services. In this context, collaboration extends beyond standard groupware facilities (e.g., floor control) and involves tracking of persistent messages exchanged in the course of synchronous collaborative sessions using semantic properties, analyzing the effect of on-line discussions and messages in terms of feedback and feed-through, as well as interaction object replication and synchronization across multiple devices with different capabilities, etc. In the course of developing initial design concepts and tentative solutions, the limitations of conventional 2D graphical toolkits were revisited in an attempt to establish a generic process and a set of strategies allowing systematic manipulation of new and diverse interaction elements. These strategies resurfaced three main topics, namely:

- the augmentation of a graphical toolkit so as to support new interaction techniques for existing / already supported (by the toolkit) interaction elements,
- the expansion of the toolkit so as to allow the creation of new and reusable interaction components and
- the integration of third-party libraries offering novel interaction facilities.

In the past, platform augmentation, expansion and integration, had been considered in the context of developing unified user interfaces capable of adapting both to the requirements of the user and the capabilities of an interaction platform (Stephanidis et al, 2001). Here, we report more recent experiences and revisit the initial concepts in an attempt to consider them as ingredients of a workflow – a process – called 'platform administration' which increasingly needs to become part of interactive software development.

3 PLATFORM ADMINISTRATION AND UI DEVELOPMENT STRATEGIES

Interaction platform administration is motivated by the increasingly pervasive nature of interactive applications (Lee et al., 2006). Its distinct aim is to establish a reusable and extensible user interface development repository (or a multiple toolkit platform) and to streamline interactive software development efforts so as to make effective use of it. Platform administration is the prime concern of environment builders and tool developers. It is an iterative process, carried out incrementally over a period of time, and seeking to establish the appropriate development environment for constructing interactive software. In this paper our aim is to discuss key activities of this process, which collectively allow for the manipulation of diverse collections of interaction objects.

Figure 1 summarizes a workflow-oriented view of this process in terms of constituent activities, outcomes, interdependencies and roles. This workflow-oriented view of platform administration could be easily revised in terms activity notation to become either a separate Rational Unified Process (RUP) workflow, or a sub-workflow embedded in the established RUP workflows.

The core theme running through the process is that user interfaces are constructed by assembling abstractions derived as a result of augmenting, expanding and integrating interaction platforms. Respectively, platform administration comprises three basic activities, namely platform augmentation, expansion and integration, which feed the activity of abstracting to compile reusable user interface development components. The term ‘component’ here implies primarily reusable class libraries, with suitable documentation (i.e. style guides) for building interactive software.

3.1 Augmentation

Augmentation involves the introduction and programmatic control of additional interaction techniques for some or all of the native interaction objects already supported by the toolkit. Augmentation is useful in cases where a toolkit’s interaction resources do not suffice to implement design concepts requiring new interaction techniques. In the past toolkit augmentation has been used to improve user interface accessibility by providing switch-based access to the Windows object library (Stephanidis et al., 1997, 2001).

However, augmentation, as discussed below, brings about usability improvements, which extent beyond disability access (Akoumianakis 2008). Figure 2 illustrates two examples of Java’s Swing augmentation of the `JTree` and `JTabbedPane` components. It should be noted that our work on augmenting the `JTabbedPane` component was carried out prior to the Java SE 6 Swing release, which supports a similar augmentation for `JTabbedPane` component. Therefore, we will briefly illustrate our approach by discussing the augmentation of the `JTree`, which is not currently supported.

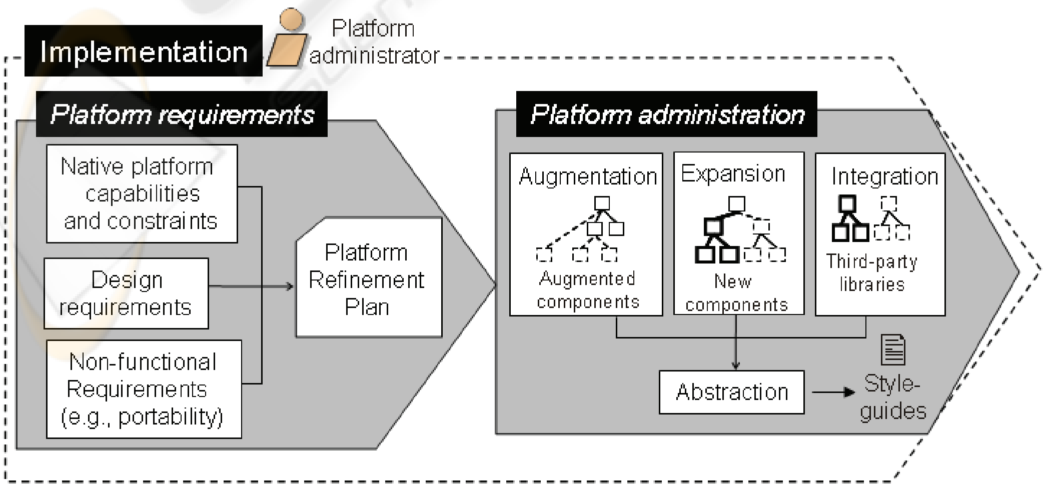

Figure 1: Platform administration process elements.

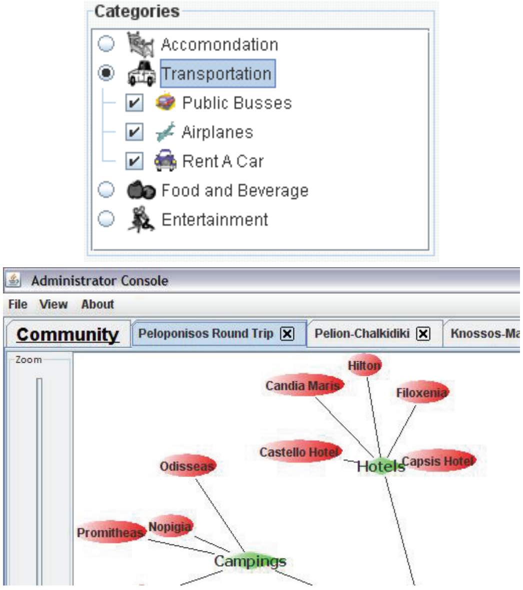

Figure 2: Example of augmented JTree and JTabbedPane.

The rationale for the augmentation arises from our intention to support both single and multiple object selection concurrently in the same component. This combined capability is not offered by any of the native Java's Swing components. Nevertheless, it is useful in cases where nested selection (i.e. pre-selection followed by multiple object selection) is required. Figure 3 illustrates the revised class model of the augmented `RadioCheckBoxTree` which supports single selection (or pre-selection) by tapping on the `JRadioButton` followed by multiple checkbox selection. It should be noted that `JRadioButton` and `JCheckBox` can be used interchangeably for pre-selection and multiple selection respectively. The augmented component also allows automatic de-selection of a parent option (`JRadioButton`) when all children checkboxes are unchecked or automatic de-selection of children when a parent option (`JRadioButton`) is unselected.

To implement the augmentation, a number of extensions to the basic Swing class library have been introduced. Specifically, `RadioCheckBoxTree` is the main class which instantiates the augmented component by delegating responsibilities to the following three classes. The class `RadioCheckBoxTreeNode`, in correspondence with JTree's default `DefaultMutableTreeNode`, is needed to hold the state of each node in relation to its type (RadioButton or CheckBox). The class `RadioCheckBoxTreeCellRenderer` is used to determine the visual appearance of the `RadioCheckBoxTree` and its components, acting as a view (in MVC terms) of each `RadioCheckBoxTree` node. The difference with the JTree's default renderer is that this custom renderer subclasses a `JPanel` instead of a `JLabel`, thus allowing hosting and presentation of visual components in addition to the classic text that a `JLabel` offers. Finally, `RadioCheckBoxMouseAdapter` undertakes the role of the controller (in MVC terms), thus tracking and propagating the user's (mouse) events and changing the model state, which in turn, delegates the event to the renderer in order to propagate modifications to the view.

3.2 Expansion

Expanding a toolkit implies the capability to introduce new domain-specific interaction objects preserving the toolkit's original programming model. Toolkit expansion is more common than toolkit augmentation. In the past it has been applied to facilitate interactive manifestation of alternative metaphors with different spatial semantics (i.e., Moll-Carrillo et al., 1995) and novel information visualization techniques (i.e., Heer et al., 2005).

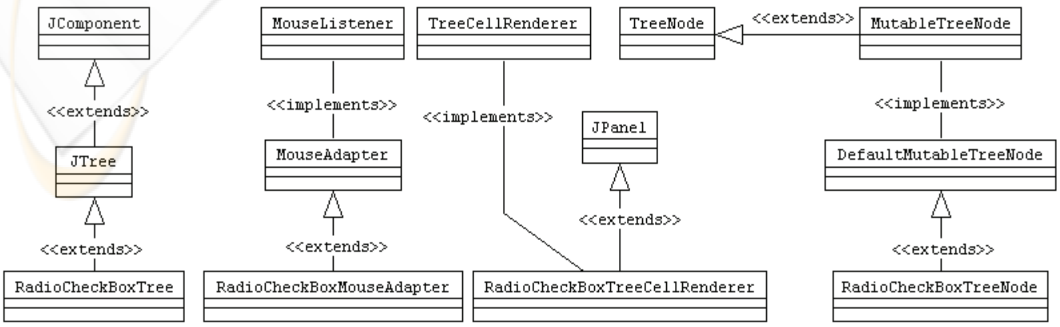

Figure 3: Swing extensions for the augmented RadioCheckBoxTree.

Moreover, expansion is the prominent strategy followed in some demonstrational user interface development techniques. We have experimented with toolkit expansion to introduce dedicated interaction components, as separate entities hosting domain-specific functionality.

Figure 4 presents an example of such a component which serves the purpose of organizing a trip by day, time and type of activity. Each activity (i.e., rectangular object) is an augmented `JButton`. The augmentation this time entails changes to the visual appearance of the object – not the behavior or interaction techniques which remain exactly as same as those of a conventional `JButton`. The figure also presents the augmented components introduced earlier, which allow creation of such buttons.

In terms of implementation, the zoom-able component `ActivityPanel` expands the Swing object library and is introduced as a new interaction component instantiated with two parameters (i.e., start date and duration). Separate objects of type `Activity` can be attached to an `ActivityPanel` using the augmented `RadioCheckBoxTree`. Each `Activity` is a selectable object realized through sub-classes of Swing's `JButton` component as shown in Figure 5.

At any time, a request for trip overview can provide a consolidated visual depiction of the entire trip as shown in Figure 4. This is obtained by a recalculation of the `ActivityPanel` so as to present each day as a column filled-up with the activities defined for that day. The resulting multi-column activity panel can be explored by zooming in and out, left and right to obtain details for a particular day and / or activity. Furthermore, the `ActivityPanel` can be easily modified so as to support additional temporal operators such as full or partial temporal overlap, containment of activities (i.e., nested activities), etc. Obviously, the approach can be further extended to allow for any type of button with different visual characteristics, as suit to the problem at hand.

3.3 Integration

Integration implies importing new interaction elements (e.g., dedicated object classes) implemented either as a separate toolkit or as a third party-library. In such a case, it is desirable the imported interaction objects to be available to the user interface developer, just as the native objects of the toolkit. It is also important to distinguish between toolkit integration as discussed here, from the multi-platform capability of existing toolkits or device-independent mark-up languages (e.g., UIML). Toolkit integration is more demanding as it assumes connectivity to arbitrary toolkits rather than a single toolkit with hard-coded implementations across different operating systems. In the context of our, we have addressed a particular aspect of integration which entails importing dedicated third-party libraries to build 2D visualizations of large volumes of data (i.e., on-line community participation, messages exchanged by participants in the course of developing a new package) and synchronization between these imported elements with conventional and / or augmented interaction components.

Figure 6 illustrates an example of integrating the JGraph visualization and layout libraries (http://www.jgraph.com/) in our running prototype to visualize messages exchanged through the eKoNEΣ message board. The distinct characteristic of this message board is that it is implemented with a dual view component. The first view makes use of `JTreeTable` to list all the messages in a hierarchical fashion within their parent topic. The second view operates on the same model to present a 2D hierarchy of messages exchanged using the JGraph Java API. The two views are interoperable and fully synchronized. Thus, when users make a choice using the 2D JGraph view the JTreeTable is automatically updated highlighting the corresponding selected item. Moreover as the JGraph view scales up or down the hierarchy of messages so does the tree-like view.

Figure 7 presents an architectural view of the current implementation of the distinct message board views. As shown, view update and synchronization is moderated by a Controller-Model abstraction which handles event traffic. This abstraction acts as an event dispatching service across the two views. Thus, when an event is dispatched, each view is notified through the eKoNEΣ controller. Views receive messages, interpret them 'locally' based on their capabilities and accordingly each view is updated. In the future, we plan to extent this basic model to allow distributed, multiple-device exploration in the context of collaborative sessions.

3.4 Abstract User Interface Components

Increasingly user interface developers face the challenge of having to program the user interface as a composition of diverse interaction components, which need not be available through a single toolkit

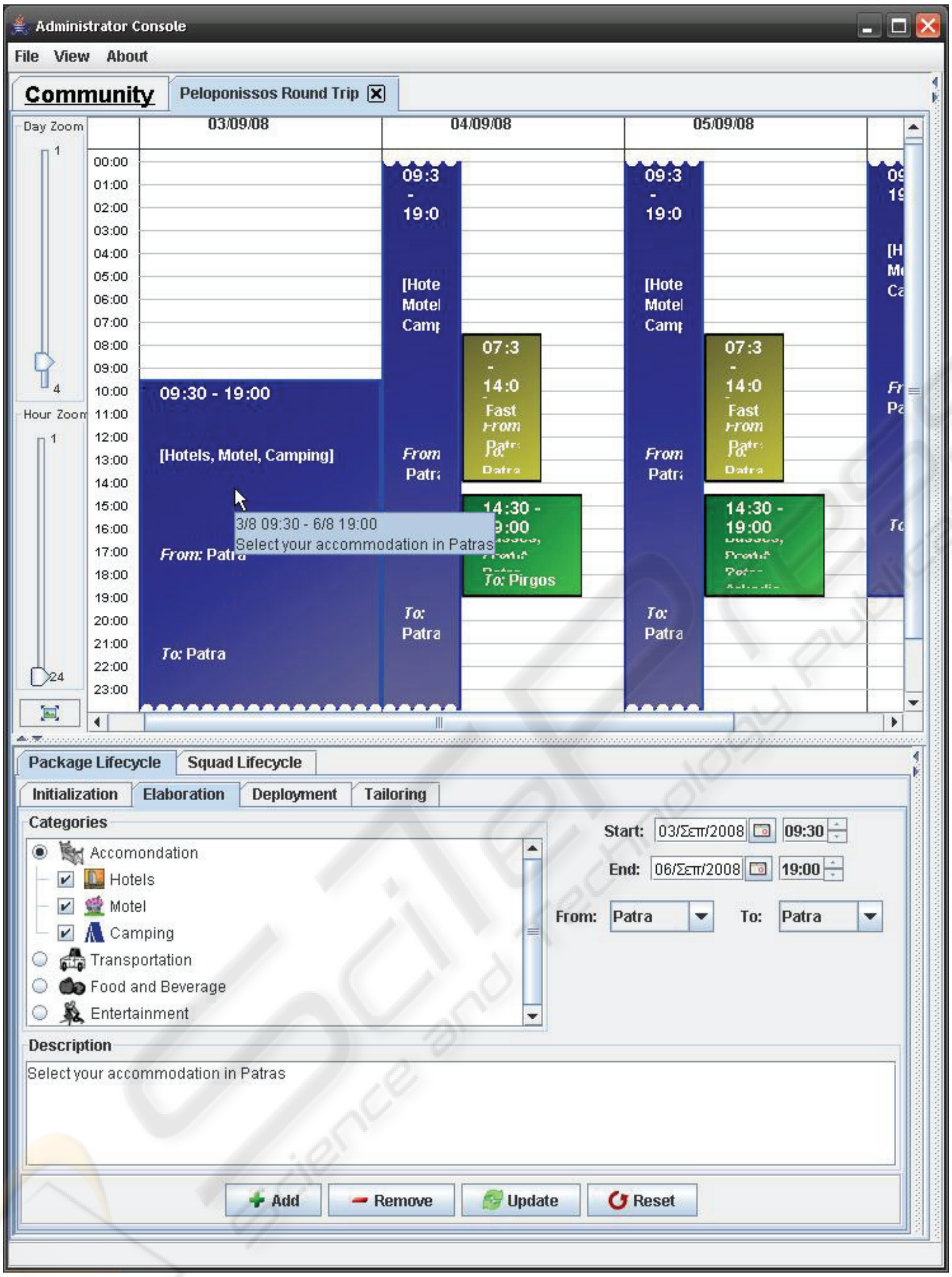

Figure 4: Example of expansion following calendar / activity organizer metaphors.

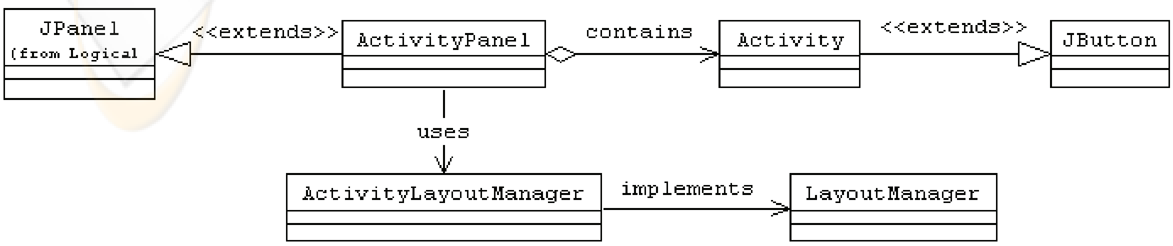

Figure 5: Swing expansion to allow the construction of activity panels hosting activities.

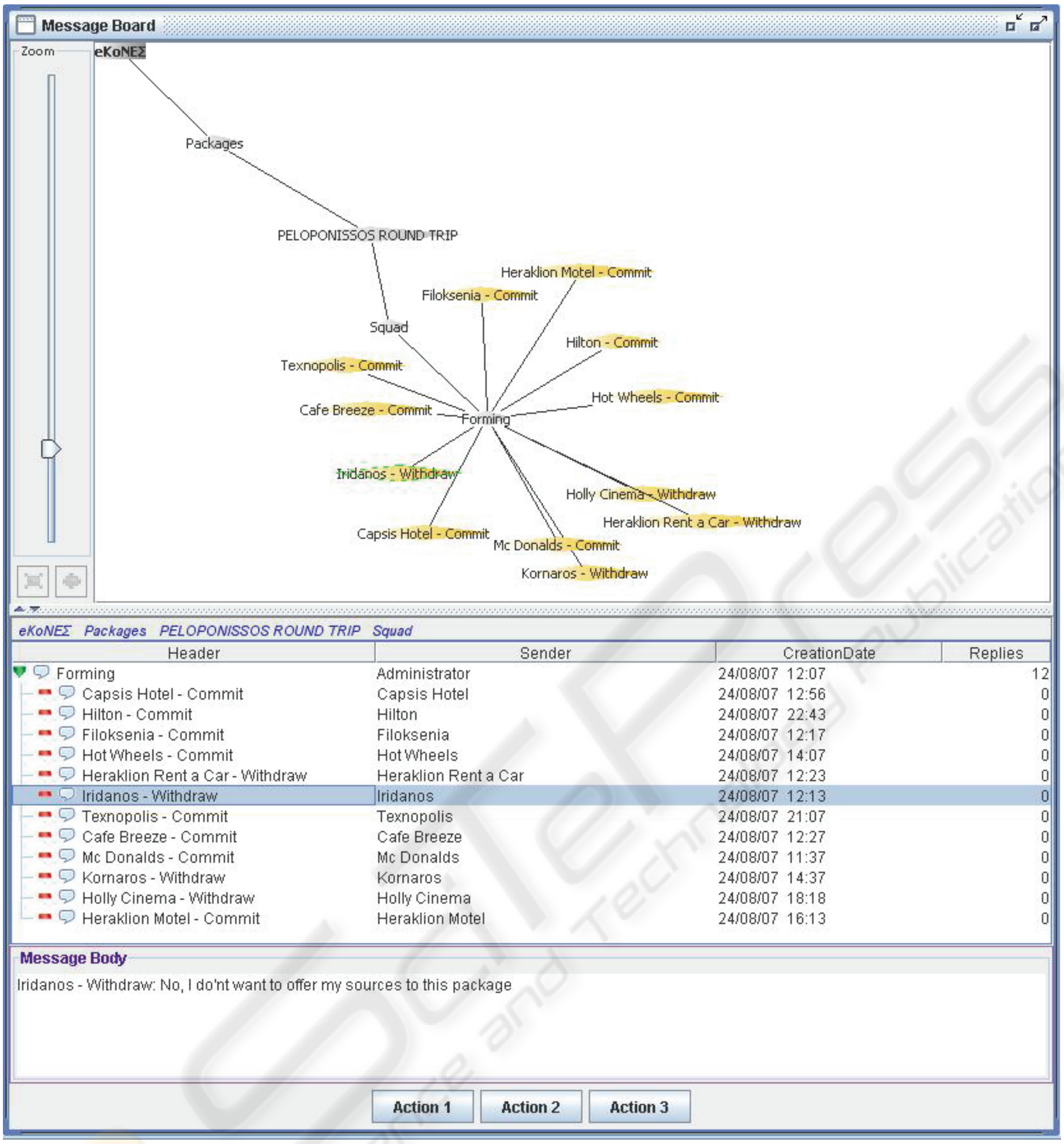

Figure 6: Example of JGraph integration.

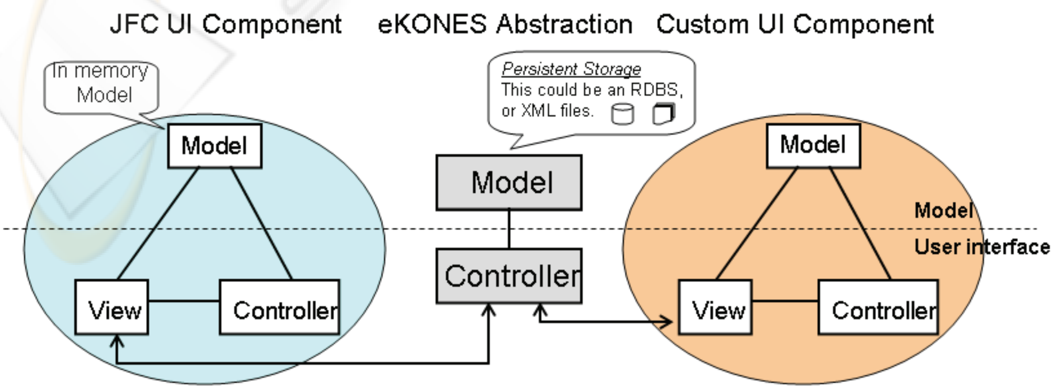

Figure 7: JGraph integration and interoperation with JTreeTable.

or interaction platform. Typically, these toolkits do not share the same programming model, which creates the need for an abstraction layer hiding toolkit-specific details and allowing ‘linking to’ rather than directly ‘calling’ each toolkit’s libraries. In previous work, we have described the Platform Integration Module which provides precisely such an abstraction layer (Savidis et al., 1997) and supports the notion of a ‘multiple toolkit platform’.

An alternative approach builds on the philosophy of separating an abstract interface description and its later rendering in any delivery context (Lee et al., 2006). The idea is that the user interface is modeled in terms of abstract elements which are then transformed to concrete instances on a target vocabulary. The model-based approach shares common ground with the notion of a multiple toolkit platform, but there are also some important differences. Specifically, the model-based approach focuses on portability, which is necessary but not sufficient to address cases where the user interface should utilize, concurrently at run-time, interaction facilities from different toolkit platforms.

4 DISCUSSION

Platform administration as presented above is typically a complex activity, seldom undertaken by tool developers. Nevertheless, it is more than likely that with the advent of new interaction technologies and the proliferation of network-attachable devices, user interface developers will increasingly need to consider some sort of platform administration. Responding to this challenge, they will increasingly need to decide what is to be augmented, expanded, developed from scratch and/or integrated. Currently, there are variable degrees of support for the strategies discussed in this paper. In particular, augmentation, although supported by most programming-based user interface development tools, it is rarely met in higher-level development tools. Expansion is also supported in most programming-oriented interface tools, but the considerable overhead, as well as the inherent implementation complexity, necessitates expert programmers. Regarding toolkit integration, the current trend is to support a multi-platform capability in a hard-coded manner (i.e., portable user interfaces using device-independent mark-up languages such as UIML). Any serious attempt to depart from the currently prevalent multi-platform capability will necessitate a more elaborate account of handling user interface abstractions.

In our recent work, all four platform administration constituents have been applied to facilitate improved interactions in the context of the running eKoNEΣ prototype, demonstrating both their potential value and technical demands. Moreover, as these strategies reflect diverse development philosophies, the paper revisited the key role of introducing and handling abstractions in the user interface development process and revisited the more demanding concept of multiple toolkit platforms. Current trends in the area of graphical toolkits (i.e. XUI, GWT, Opera widgets) indicate the wide range of interaction components available to designers/developers and motivate the need for multiple toolkit platforms.

5 SUMMARY AND CONCLUSIONS

This paper has presented the ingredients of a user interface development process aiming to advance techniques for manipulating diverse collections of interaction elements. The process entails four distinct constituent activities, namely augmentation, expansion, integration and abstraction, likely to be relevant when the implemented libraries of a designated toolkit / platform do not suffice.

Augmentation refers to introducing new interaction techniques for already supported interaction objects. Augmentation is relevant in cases where input interaction techniques such as mouse selection or output behaviours (i.e. look and feel of an interaction object) need to be revised to suit a particular situation. For instance augmentation is appropriate for introducing accessible interaction techniques (i.e., switch-based access for motor-impaired users) to graphical objects. In this paper we have also shown how it can be used to allow nested selections (i.e. pre-selection followed by selection as indicated in Figure 3). On the other hand, expansion entails the capability of constructing new interaction elements either as generic or domain-specific components. This is useful when articulating elements of a domain-specific visual language or when introducing alternative interaction metaphors (see the activity panel in Figure 4). Integration allows importing interaction components realized as third-party libraries. The problem here is synchronizing views compiled using objects classes of two or more different class libraries (see Figure 6). To this end, we have described how abstract components (model-controller combinations) can be used to synchronize 'local' views of the same model. This leads to the issue of manipulating abstractions to enable synchronization of augmented, expanded and integrated interaction elements.

ACKNOWLEDGEMENTS

The present work is carried out in the context of the eKoNEΣ project (KP-25). The project is co-funded by the General Secretariat for Research and Technology, Greek Ministry of Development, in the context of the Operational Programme of the Region of Crete (PEP-Crete), Activity 1.2 – 'Research and Technological Development consortia in areas of strategic priority'.

REFERENCES

Akoumianakis D. (2008, in print): Managing universal accessibility in software-intensive projects, *Software Process: Improvement & Practice*, Vol. 13.

Akoumianakis D., Vidakis N., Vellis G., Milolidakis G., Kotsalis D. (2008): Interaction scenarios in the 'social' experience factory': Assembling collaborative artefacts through component reuse and social interaction, *Proceedings of the 3rd IASTED International Conference on Human Computer Interaction*, March 17 – 19, Innsbruck, Austria.

Akoumianakis, D., Stephanidis, C. Multiple Metaphor Environments: designing for diversity, *Ergonomics*, 46 (1-3), 2003, 88-113.

Akoumianakis, D., Vidakis, N., Vellis, G., Kotsalis, D., Milolidakis, G. Experience-based social and collaborative performance in an 'electronic village' of local interest: The eKoNEΣ framework, *Proceedings of 9th International Conference on Enterprise Information Systems (ICEIS'2007)*, 12-16 June, Madeira - Portugal.

Bederson, B. B., Grosjean, J., Meyer, J. Toolkit Design for Interactive Structured Graphics, *IEEE Transactions on Software Engineering*, 30 (8), pp. 535-546, 2004.

Bederson, B.B., Meyer, J., Good, L. Jazz: An Extensible Zoomable User Interface Graphics Toolkit in Java, *Proceedings of ACM \Workshop on User Interface Software and Technology (UIST' 2000)*, pp. 171-180 2000.

Heer, J., Card, S., Landay, J. prefuse: a toolkit for interactive information visualization, *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'05)*, April 2–7, 2005, Portland, Oregon, USA.

Lee, C., Helal, S., Lee W. Universal Interactions with Smart Spaces, *IEEE Pervasing Computing* (January-March), 2006, pp. 16-21.

Moll-Carrillo, H.J., Salomon, G., March, M., Fulton Suri, J., and Spreenber, P. Articulating a Metaphor through user-centred design, *Proceedings of ACM CHI'1995*, Denver, 1995, 566–572.

Perry, M., O'hara, K., Sellen, A., Brown, B., Harper, R. Dealing with Mobility: Understanding Access Anytime, Anywhere, *ACM Transactions on Computer-Human Interaction*, 8 (4), 2001, 323–347.

Savidis A, Stephanidis C, Akoumianakis D., Unifying toolkit programming layers: a multi-purpose toolkit integration module. In Harrison MD, Torres JC (eds) *Proc. of the 4th DSV-IS'97*, Granada, Spain, 4–6 June 1997. Springer, pp 177–192.

Stephanidis C, Savidis A, Akoumianakis D. Universally accessible UIs: the unified user interface development, Tutorial in *ACM Conf. on Human Factors in Computing Systems (CHI'01)*, Seattle, USA, 2001.

Stephanidis, C., Savidis, A., & Akoumianakis, D. Unified Interface Development: Tools for Constructing Accessible and Usable User interfaces, *Tutorial no 13 in the 7th HCI International '97*, San Francisco, USA, 1997.