Specification for Visual Requirements of Work-Centered Software Systems

James R. Knapp
Department of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435
knapp.12@wright.edu

Vincent A. Schmidt
Air Force Research Laboratory
AFRL/HECS
Wright Patterson AFB
vincent.schmidt@wpafb.af.mil

Soon M. Chung
Department of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435
soon.chung@wright.edu

Abstract

Work-centered software systems function as inherent work-aiding systems. Based on the design concept for a work-centered support system (WCSS), these software systems support user tasks and goals through both direct and indirect aiding methods within the interface client. In order to ensure the coherent development and delivery of work-centered software products, WCSS visual requirements must be specified to capture the cognitive aspects of the user interface design. A work-centered specification language based on the User Interface Markup Language (UIML) is an effective solution to bridging this gap between cognitive systems engineering and software engineering. In this paper, we propose a new visual requirements specification language that can capture and describe work-centered visual requirements within a semi-formal syntax. The proposed language can also be easily integrated into a UML object model via the use of UML's extensibility features. A specification language for visual requirements could be employed by cognitive engineers and design teams to help convey requirements in a comprehensible format that is suitable for a software engineer. Such a solution provides coherency in the software modeling process of developing work-centered software systems.

1. Introduction

Every new software system requires a different and unique software process to best suit its individual development needs. A major milestone common to nearly every software process is the specification of the product to be developed. This entails the definition of the software's operation as well as constraints upon this operation. As a crucial first step in the software modeling process, specification has direct influence upon development, implementation, and further progress. A good software process is one which reliably communicates information from one step to another, laying a foundation for coherence across the entire set of development iterations.

There exists a specific type of software system known as a work-centered support system (WCSS). Work-centered support systems possess a large number of cognitive and visual requirements associated with the software user interface. When these systems are specified, it is common that many of these valuable requirements are lost throughout development due to their difficult visual (vs. functional) nature. Without the potential to communicate how requirements and design intent interconnect, the probability of creating a stable, adaptable, and coherent WCSS in the software product is significantly diminished. This issue is often intensified by the fact that software personnel rarely have knowledge of the actual work context from which the design was developed. By creating a notation to specify visual requirements, as well as showing capability for integrating with traditional software modeling techniques, this paper lays groundwork for the establishment of a software process tailored to capture work-centered software requirements. In this paper, we present a potential solution to this eminent issue of specifying visual requirements through the use of a XML-based specification language.

The Unified Modeling Language (UML) has shown itself to be an effective method employed in the software modeling process. The UML provides a versatile starting point for covering the various facets of a software development process. The premise for the creation of the UML was fundamentally communication [1]. Without a standard by which engineers and non-systems personnel could effectively communicate, the growth of the field of software development was considerably handicapped. The UML possesses valuable assets in being able to continue to grow and extend its capabilities to meet new software development needs. Additions such as the upgrade to UML 2.0 give strength to the UML's versatility and show that it has potential for the long term.

Even with UML modeling capabilities, the amount of data presented to the user in new software products continues to grow as software is developed to accomplish more complex work tasks. Along with total data, the level of computational complexity for the user is also rising rapidly. Software users must perceive, absorb, and make more complicated decisions within virtual environments than ever before. Government agencies such as the Department of Defense are migrating towards net-centric environments where many various information repositories are linked together. Net-centric environments, therefore, are likely to further exacerbate the information overload problem. This approach and others like it do not ensure that appropriate views of the filtered data are relevant to support work or give any initial work environment representation which can then be customized and tailored to the work being done [2]. In order for software to be fully successful in military, business, and other applications it must be functionally adept to accomplish its tasks, as well as helpful and convenient to its end-users in completing those tasks.

Developing software systems with a focus on work to be accomplished has been researched and defined by Eggleston et al. as a work-centered support system [5]. According to their definition, a work-centered support system design approaches the work representation in terms of how workers see and engage work [3]. This work representation effectively captures the work ontology, which is essential for building software around the work environment. Within this theory, software is developed under a work-oriented framework, allowing components such as the software interface client to be developed as a work support aid. As a system is conceptualized, it is made to implicitly support the user in completing work.

From a work-centered point of view, the UML shows inadequacy in its ability to model all aspects of work-oriented behavior. Although a powerful tool, the UML has no specific modeling of user goals and intentions, showing it to be incapable of expressing user and usage-oriented functionality [4]. The UML was not designed to be an all-encompassing modeling tool, and displays an overall lack of support in the development of a complete work-centered object model. In general, UML methods are relatively informal, emphasizing usefulness rather than precision. The UML serves to highlight the important details and retain the most desired features in the development of a system. However, since UML is the primary software engineering development tool used by current systems developers, this deficiency in expressing usage-oriented functionality keeps many systems from being designed in an optimal work-centered manner. This gap that exists in relation to the UML correlates to a more abstract need for a linkage between software engineering and cognitive systems engineering in this area. It is important to note that while the UML is not all sufficient it does retain the possibility for further extensions and enhancements, and even encourages such augmentation. Many experts seem to agree that existing gaps could be bridged by making alterations to the already standard and robust mainstay that is the UML [4]. The specification language proposed in this paper exploits this attribute in the creation of a notation which can effectively integrate with the UML.

Current software engineering processes and methods have difficulty in expressing certain types of requirements throughout project development. Many of these are known as non-functional requirements. These types of requirements often involve factors closely associated with user work. The user interface (visual portion) of the software product falls firmly into the category of being supportive to work tasks. The UML does not currently support this representational layer of the design. For work-supporting software to become a reality, the ability to specify work-centered requirements from a software engineering perspective must be made available.

As no specification exists currently for formally capturing work-centered requirements as an aid to software design, this paper proposes a work-centered specification language based on the User Interface Markup Language (UIML) as an effective solution to bridging this gap between cognitive systems engineering and software engineering. The proposed language is described in terms of formal syntax, semantics, and employment to capture work-centered visual requirements. Along with this description, this paper also elaborates on how the proposed language can be easily integrated into a UML object model via the use of augmentation features already available within the UML. This paper covers all relevant aspects necessary to understand how this specification language is a generous contribution to coherently modeling work-centered software systems.

2. Background

2.1. Work-centered support systems

A work-centered support system (WCSS) is a fully integrated support system focused on helping the user accomplish work. As defined by Eggleston et al., a WCSS appears as a graphical user interface with embedded support tools in a work-centered organizational structure [5]. WCSSs are based on the idea that the primary purpose of the user interface is to function as a work aiding system. WCSS combines representational aiding with intelligent automation within a single organizing framework [6]. WCSS is both a design technology and an interface client technology for the user interface layer of software application [5].

As an interface client technology, WCSS dictates portions of control structure, object model, and user interface in a software product [5]. In normal practice, the form of the user interface is dominated by concerns over information object design, incorporation of good human factors, and meeting general style guidelines for human-computer interaction. However, little effort has been devoted to treating the interface as a support system in its own right [3]. The WCSS approach achieves effective support for these cognitive concerns by blending various aiding tools in a manner that is tailored to the characteristics of user work [5]. In this form, the speed and quality of task decision making is improved and the amount of cognitive burden placed on the user is minimized. By highlighting and representing the key features of the work domain, the interface is sensitive to the work context and able to support the range of work assigned to the user. The use of work domain visualizations and common work terms function as indirect work support aids [7]. The main ingredients which constitute a work-centered support system are: a set of representational forms that themselves act simultaneously as work aids and GUI panels (for perceptual-based analysis and situational awareness), a set of different classes of software agents crafted and made available to automatically perform work tasks under the guidance and control of the user, and a common work ontology to connect the various forms of aiding.

Representational forms present work tasks in domain terms, and show the problem state, environment constraints, and resources available for their completion [6]. These types of aids are context relevant, meaning they attempt to capture the work domain instead of simply being an activity-based model. When the work domain is used as a base point for providing support, the aid better accommodates the flexible and adaptive nature of user work [6]. Representational forms are supplied by the graphical and visual portions of the user interface. This includes the necessary components for the work domain to be represented within the software support tool.

A key tenet in the success of the development of a work-centered software system is the notion of coherence. The sustainment of vital work-centered details must occur from elicitation and knowledge capture to implementation and final development to certify a successful work-centered software system which fully realizes the WCSS design paradigm. Transmission of fundamental design artifacts is indispensable in order to prevent the destruction of the designed work-centered orientation at development trade-off points. This includes the transfer of the final work-centered design synthesis to software engineers and developers for final implementation. Without a medium for communication in the final design stages, the entire work-centered design is at risk.

2.2. Timeline Tool: a work-centered support system

The Timeline Tool is a concrete example of a WCSS being developed by the Department of Defense [8]. Its inception and design was in response to needs of the Air Mobility Command Tanker Airlift Control Center (AMC TACC). The TACC schedules and tracks strategic tanker and airlift resources worldwide. Air Force and Department of Defense support taskings are carried out at the TACC by several hundred people planning and executing around 350 missions per day [8]. However, within the current systems being used to support the airlift planning, personnel suffer from not always having the correct data displayed in a suitable format at the right time.

The activity of mission planning is both dynamic and complex, thus the control center staff must synthesize a plethora of factors during this process including: passenger and cargo logistics, diplomatic clearance permissions (DIP), payload information, flight plans, air refueling, Notices to Airmen (NOTAM), and weather forecasts generated by meteorological specialists.

To add further difficulty, the many sources of data required to effectively plan airlift missions are commonly segregated from other corresponding and relevant information. In order to make sense of the valuable data, the officer on duty must piece together the data from all its various locales. It is then quite common that the operator must “mentally fuse” or transform the data in off-line calculations to arrive at the needed value or format to support effective decision making [8]. All of these preliminary activities result in the duty officers having a high amount of cognitive and decision-making burden as they go about their daily planning activities. Currently, officers utilize multiple large monitors displaying enormous amounts of raw data in spread-sheet formats, as well as needing to retrieve data from additional sources manually via other personnel [8].

The Timeline Tool, when fully developed as a work-centered software system, will alleviate a significant amount of the current hardships in airlift mission planning. This is a direct result of both automated and representational support aids being integrated into the software system. The additional software agents which automate important work-aiding computations as well as the visualizations and user interface portions of the system have been designed by a team of cognitive and work-centered experts. The new design fuses and correlates the many sources of data relevant to mission planning together into a well-formed representation of the work context within the user interface.

Inside the new representational framework, vital mission information will be collected into a "core" display for each mission. A comprehensive multi-mission window will allow officers to monitor the pivotal details of many mission cores simultaneously. Each individual mission unit can also then be localized into a "core and clusters" view, which shows the related topical clusters (port availability, crew schedules, etc.) and how they interrelate to flight plans. The Timeline Tool's list of benefits and advantages over current legacy systems is extensive and will further propagate the work-centered paradigm as a profitable design method.

3. A specification language for work-centered visual requirements

Creating a specification language that is able to encapsulate the important aspects of the visual design portion of a WCSS will allow coherent work-centered software systems to be created. Such a specification creates a framework for the transmission of visual aspects of design in a precise and unambiguous fashion. Additionally, the language enables flexibility and concision in the selection and labelling of explicit design elements. Most importantly, a specification language for visual requirements would be used by cognitive engineers and user interface designers, those who are most familiar with the work-centered design, to convey the important design artifacts to software engineers. Thus, essentially it has the same purpose and goal of UML: to provide a medium of communication to talk about software system modeling.

In order for a specification language for visual requirements to be truly beneficial, it must also integrate well with existing software engineering modeling techniques. Using a language as a standalone requirements specification amid other gross functional requirements and modeling documents will only increase its chances of being overlooked during development. An effective modeling document is one in which the system design can be best represented in its entirety, as not to additionally burden the development team when they proceed to build the software. This motif of "having everything in one place" thus implies that a specification language for visual requirements should integrate well into a standard modeling language such as the UML. The UML makes this quite feasible through its natural extensibility. Using the UML's existing outlets for connecting outside modeling techniques will provide the needed linkage for a visual requirements specification.

3.1. Employing the User Interface Markup Language (UIML) as a language framework

The User Interface Markup Language (UIML) is an XML compliant meta-language for describing user interfaces. Its documentation reads, "the design objective for the UIML is to provide a canonical representation of any UI suitable for mapping to existing languages." [9] Typically, UIML is used to describe generic "window-oriented" user interfaces and is then passed into an interpreter which implements the design into a higher-level programming language such as Java or C++. Unfortunately, these simplistic designs are not frequently applicable to unique and complex UIs such as those of a WCSS. However, the canonical representation still possesses many advantages in specifying a more intricate UI. The UIML language can be modified to capture more complex designs with the goal of transferring them to a human development team, instead of to a machine interpreter. These enhanced UIML design files can then be incorporated with other design documents to provide a uniform design model. The set of modifications which transform the UIML into a visual requirements specification language are described in this section. Beforehand, it is important to note several other useful features and attributes of the UIML.

The UIML shows usefulness for specification in its decentralized and scalable structure. When employed, the UIML divides a UI design into a set of unique logical parts or objects. This enables the designer to break down a custom UI and specify each part at an appropriate level of detail. Each piece is able to contain a set of nested child parts, which can then be labeled and specified within the context of their parent. The specification proceeds recursively in this outline format until all the required details at the lowest level are captured. In association with each part's specification, the UIML uses a toolkit vocabulary to keep track of the various part types. This vocabulary is established by the designer to effectively label each class of part for correct identification. Using these abilities to scale and effectively decentralize design requirements, a designer can define unique user interfaces with precision and clarity at the individual part level in order to maintain a work-centered domain focus.

3.2 Modifying the UIML to create a visual requirements specification language

In its current form, the UIML is a suitable basis for a specification language, but still lacks several important characteristics to make it effectual for WCSS visual requirements. By modifying the existing UIML, we can make use of its features for formal syntax, canonical form, decentralized and scalable structure, and part vocabulary toward the goal of conveying a UI design to a development team. First, the UIML's features for part description and definition must be made comprehensive enough to merit a clear specification of complex design artifacts. This deficiency is addressed by the addition of attribute tags which replace the current tag categories. Second, since the language will be employed as a human-to-human protocol rather than human-to-machine, a more understandable formatting is needed. In order for human factors scientists and other user interface specialists to effectively use the proposed language, it must be represented in a non-code-like syntax. These issues and other minor additions are achieved by modifying the UIML to create a new visual requirements specification language.

The original UIML syntax is composed of four major elements used to describe each part of the user interface. The four element tags are: *structure*, *style*, *content*, and *behavior*. Each part is defined in terms of these four major elements. The visual requirements specification language makes use of these elements in capturing requirements. However, additional and modified tags and features are given to replace those of the original UIML, where further description is necessary. These additions equip the new language to better encapsulate each part's requirements.

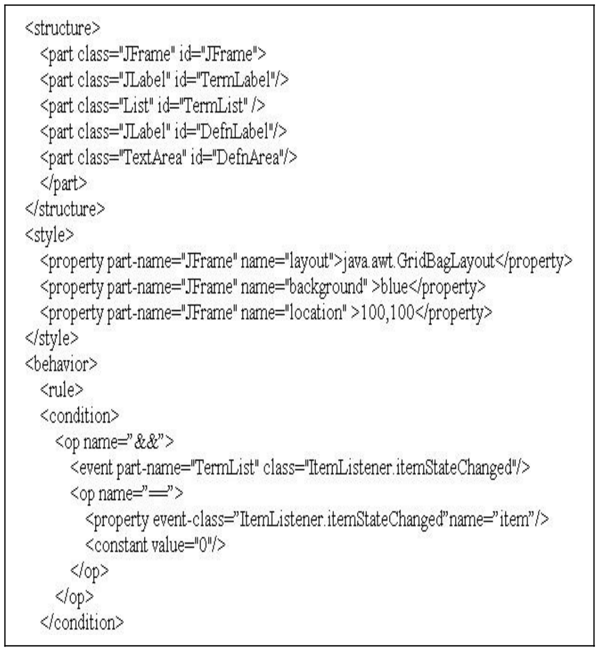
A visual requirements language's usage of the *structure* tag to identify parts coincides with that of the standard UIML. Each part is given a unique identifier (composed of alpha characters) which can then be used as a reference. A class name is also associated with each identifier to properly define the part type. This class definition is the primary vehicle for the developer to understand why this part of the UI exists and is important in the WCSS. Each class is defined in the external toolkit vocabulary, providing design rationale and universal class details. Subsequent parts are similarly defined in a recursive outline format, making use of nesting to correctly identify subsumed parts.

The *style* and *content* tags are used by the UIML to capture the presentation details of each UI part. Unfortunately, WCSS visual requirements often contain much more varying aspects to be captured than simply those of style and content. This includes attributes such as location, size, and formatting rules. To encapsulate these more definitive features, style and content are incorporated into a new and broader tag category named *static attributes*. As many of these attributes as necessary can be included for any of the aforementioned particulars of an individual part.

The *behavior* tag for each part defines how it should react to certain conditions and circumstances during the life of the user interface. Since many WCSS UI parts may have more than one type of behaviour in any given case, the *behavior* tag has been replaced by a more customized *dynamic attributes* tag in the visual specification language. Under this new heading, multiple specific dynamic events can be identified as sets of event/action pairs. Each event listed under a dynamic attribute constitutes a triggering condition for the corresponding action. When such an event is triggered (e.g. a button is clicked), the expression listed under the action clause will be executed (e.g. load a new page). Each event/action pair is therefore able to capture one dynamic attribute of behavior. In order to cater to behavior more complex than can be expressed by simple Boolean expressions, a keyword "call" is introduced. The *call* keyword exists to create an external functional reference for modeling complex behavior. In this way, dynamic actions based upon events such as mouse location, time of day, or numeric computations can be observed.

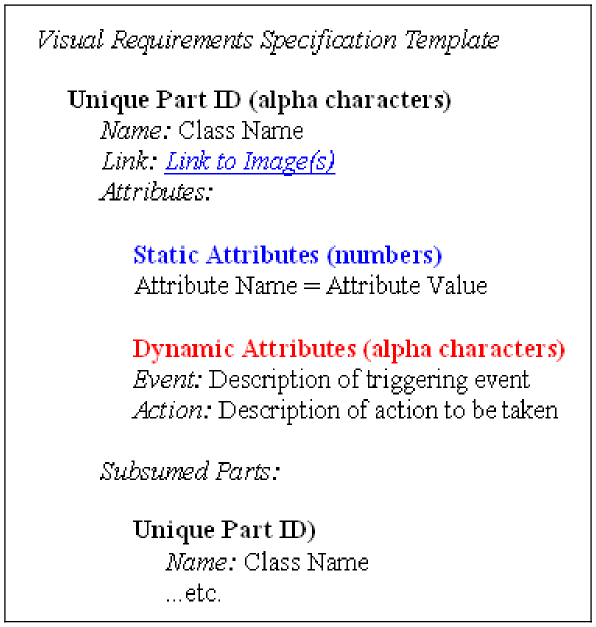
In addition to these standard modules, a linking module is introduced to allow a mechanism for communicating additional part information. The *link* tag provides a way to associate a part with other relevant items of the specification for greater coherency. Such items could include: another document, an image, a video file, an interactive prototype, or anything else that might be vital to the comprehension of the UI part.

In order to make the visual requirements language comprehensible, it must be made visually appealing to its users. This goal is met by replacing the standard XML tags for each module with plain prose, as well as the addition of simple color-coding. The resulting format is both more readable and easier to create using a simple word processing application. From a technical standpoint, each document is still easily transferable into an XML format as each has distinct module names and an outline format. Upon completion of the specification, the UI designer can forward the documents to the functional design team who can then incorporate them into the overall project design which will be sent to the developer. One such method for doing this would be to use the UML integration proposed in Section 3.4. A portion of a standard UIML document, a specification template for the new visual requirements language, and a mapping between the two are given in Figures 1 to 3.



```
<structure>
  <part class="JFrame" id="JFrame">
  <part class="JLabel" id="TermLabel"/>
  <part class="List" id="TermList" />
  <part class="JLabel" id="DefnLabel"/>
  <part class="TextArea" id="DefnArea"/>
  </part>
</structure>
<style>
  <property part-name="JFrame" name="layout">java.awt.GridBagLayout</property>
  <property part-name="JFrame" name="background" >blue</property>
  <property part-name="JFrame" name="location" >100,100</property>
</style>
<behavior>
  <rule>
  <condition>
    <op name="&&">
      <event part-name="TermList" class="ItemListener.itemStateChanged"/>
      <op name="==">
        <property event-class="ItemListener.itemStateChanged"name="item"/>
        <constant value="0"/>
      </op>
    </op>
  </condition>
```

Figure 1. Sample of a standard UIML document



Visual Requirements Specification Template

Unique Part ID (alpha characters)
Name: Class Name
Link: *Link to Image(s)*
Attributes:

Static Attributes (numbers)
Attribute Name = Attribute Value

Dynamic Attributes (alpha characters)
Event: Description of triggering event
Action: Description of action to be taken

Subsumed Parts:

Unique Part ID)
Name: Class Name
...etc.

Figure 2. Visual requirements specification template

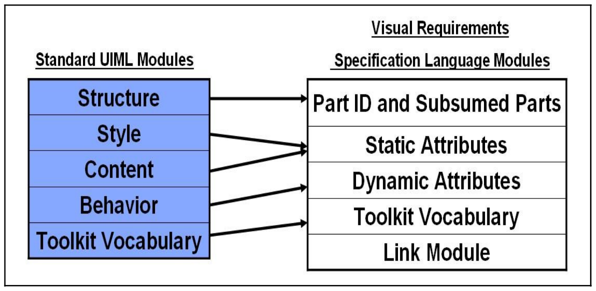


Figure 3. UIML to the visual requirements specification language mapping

3.3 Applying the visual requirements specification language to the Timeline Tool

The new specification language is applied to the Timeline Tool as an example to give a picture of what a full visual requirements specification might contain. In this section, we focus on the Airfield Cluster portion of the Timeline Tool. This cluster displays information pertaining to ports and airfields where aircraft may debark, land, or refuel. The original design drawings for the cluster are used in Figures 5 to 7 as a link reference.

To use the new specification language, we start by dividing the cluster logically into a set of parts. Due to space limitations, only a portion of that division is given. The specification for the selected parts is given in Figure 4.

BBC)
Name: Airfield Cluster
Link: *Airfield-Cluster*
Attributes: None
Subsumed Parts:

BBCB)
Name: Central Airfield Timeline
Link: *Airfield-Central-Timeline*
Attributes:

1) Height = 100 pixels
2) Width = 520 pixels

a) *Event:* Mouse Over Any Subsumed Part
Action: Display Time Parameters Tooltip
b) *Event:* Alert/Violation
Action: Change Background Color to Alert Status
d) *Event:* Airfield Timeline Display Region
Action: **Call** Get Projected Arrival Times

Subsumed Parts:

BBCBA)
Name: Individual Timelines
Link: *Individual-Timeline*
Attributes:

a) *Event:* Content Data Updates
Action: **CALL** Retrieve External Timeline Data

Subsumed Parts: 5
etc…etc.

Figure 4. Airfield cluster specification

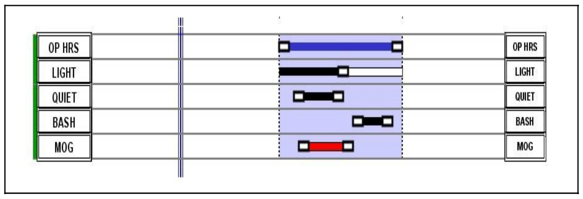


Figure 5. Image of full airfield cluster (part BBC)

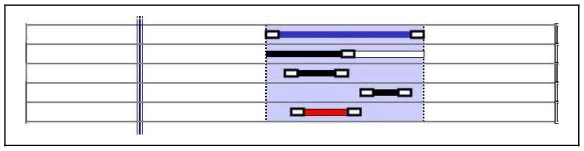


Figure 6. Image of central airfield timeline (part BBCB)

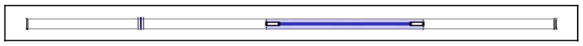


Figure 7. Image of individual timeline

(part BBCBA)

Within the specification in Figure 4, three parts are specified. Images of each of these three parts are given in Figures 5 to 7. The first part is identified as part BBC. The class of this part is listed as "Airfield Cluster". This class name corresponds to an entry within the external toolkit vocabulary document which would further elaborate on work-orientation and design rationale for this particular class. Below the class name, a link is included to an image of this part (Figure 5). This image gives the software developer a window of understanding for what is being specified on this particular part. In this case, no attributes are specified at this highest part level, but instead are left to be given at the level of each subsumed part. Part BBCB is subsumed within part BBC. Its declaration is identical in syntax to that of part BBC. Here, several attributes have been provided, including visual dimension static attributes, and mouse over, alert, and display region dynamic events. Inside of this part declaration, the third part, BBCBA, is nested. At this lowest level, each individual timeline is attributed to retrieve its individual data feed. The nesting syntax allows distinction between each timeline at the lowest level of detail, while maintaining that each adheres to all parent attributes.

The specification hereby provides the concrete visual requirements necessary for this portion of the software interface to be developed according to a work-oriented framework. It can be assumed that details for these three interface pieces, which are above and beyond those given within the specification, are irrelevant to the work-centered focus.

The visual requirements specification language captures the valuable information and provides a channel for communicating it effectively to the developer, ensuring a function work-centered software system as a result. In contrast, using current approaches (with only UML) leave behind valuable visual design requirements. These details, such as the combined depiction of the various data stores of airfield information, comprise the work-centered software representation which allows the functional application to meet the user's need in an optimal manner. Without specification of such details, it is unlikely that any work-centered software system can be accurately realized in a non-specialized development setting.

3.4 Integrating the visual requirements specification language with the Unified Modeling Language (UML)

In order to be applicable to standard software engineering practices, the new specification language must integrate well with existing software design methodologies. This is quite easily done via the UML's various facets for extension. In Figures 8 and 9 are given two possibilities for integration: using the UML package notation to group functional (UML) specification and related visual requirements specification together; and using the UML comment notation to include links to visual specification at appropriate design points. A newly devised visual specification language complements the UML with addition capability for capturing the overall system design, both process and presentation.

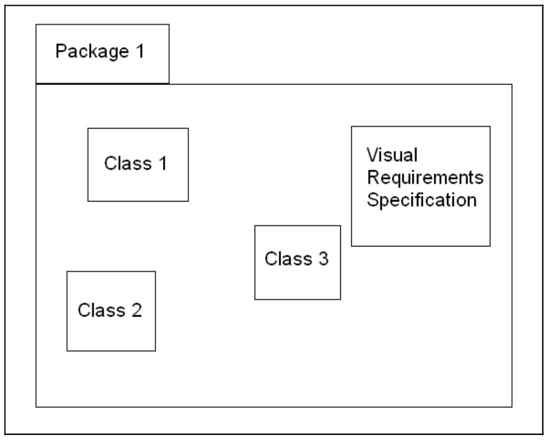


Figure 8. UML and visual requirements package integration

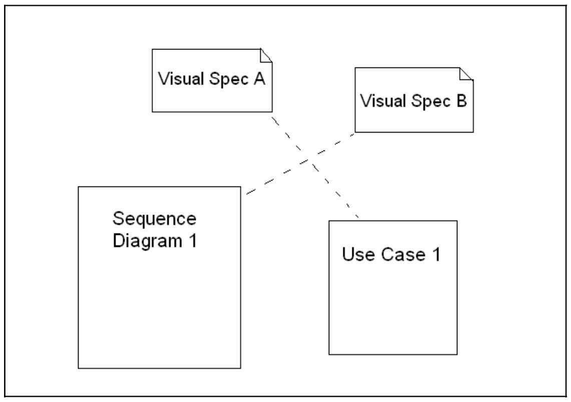


Figure 9. UML and visual requirements comment integration

4. Conclusion

A work-centered specification language based upon the User Interface Markup Language (UIML) is an effective solution to capturing complex and intricate visual requirements of user interfaces, for which there is currently no existing tool or method to obtain. This new visual requirements specification language can bridge the gap between design and development for future work-centered software systems, preventing the loss of vital work-centered visual requirements at development trade-off points. The language provides a method by which UI designers can formally capture WCSS visual design elements, maintaining the work-centered focus which is essential during software development. As an additional utility for capturing visual requirements, the new language can also be integrated with the UML to provide a more complete requirements model.

Acknowledgment

This research is supported in part by a research fellowship jointly provided by the Air Force Research Laboratory and the Dayton Area Graduate Studies Institute.

References

[1] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd edition, Addison-Wesley, 2004.

[2] J. Wampler, "Work-Centered Software Engineering," AFRL/HECS, Dayton, Ohio, 2004.

[3] R. G. Eggleston and R. D. Whitaker, "Work-Centered Support System Design: Using Frames to Reduce Work Complexity," Proc. of the 46th Annual Meeting of the Human factors and Ergonomics Society Meeting, 2002.

[4] C. Rourke, "Making UML the Lingua Franca of Usable System Design," *Interfaces Magazine*, 2002.

[5] R. G. Eggleston, M. J. Young, and R. D. Whitaker, "Work-Centered Support System Technology: A New Interface Client Technology for the Battlespace Infosphere," Proc. of IEEE Nat'l Aerospace and Electronics Conf., 2000, pp. 499-506.

[6] R. G. Eggleston, "Combining Representational and Automation Methods to Aid Complex Work," *Proc. of Sym. on Analysis, Design, and Evaluation of Human-Machine System*, 2004.

[7] R. Scott, E. Roth, S. Deutsch, E. Malchiodi, T. Kazmierczak, R. Eggleston, S. Kuper, and R. Whitaker, "Work-Centered Support Systems: A Human-Centered Approach to Intelligent System Design," *IEEE Intelligent Systems*, Vol. 20, No. 2, 2005, pp. 73-81.

[8] J. Wampler, R. Whitaker, E. Roth, R. Scott, M. Stilson, and G. Thomas-Meyers, "Cognitive Work Aids for C2 Planning: Actionable Information to Support Operational Decision Making," Proc. of *10th Int'l Command and Control Research and Technology Symposium: The Future of C2*, 2005.

[9] M. Abrams and J. Helms, "User Interface Markup Language Specification Working Draft 3.1," www.oasis-open.org/committees/documents.php?wg_abbrev=uiml, March, 2004.