

# Journal of International Technology and Information Management

---

Volume 14 | Issue 3

Article 4

---

2005


## UML Activity Diagram Semantics and Automated GUI Prototyping

Nancy Winniford Ashley  
*Washington State University, WA*

Timothy E. Meehan  
*Nuvotec, Inc., WA*

Norman Carr  
*Nuvotec, Inc., WA*

Follow this and additional works at: <http://scholarworks.lib.csusb.edu/jitim>

 Part of the [Business Intelligence Commons](#), [E-Commerce Commons](#), [Management Information Systems Commons](#), [Management Sciences and Quantitative Methods Commons](#), [Operational Research Commons](#), and the [Technology and Innovation Commons](#)

---

### Recommended Citation

Ashley, Nancy Winniford; Meehan, Timothy E.; and Carr, Norman (2005) "UML Activity Diagram Semantics and Automated GUI Prototyping," *Journal of International Technology and Information Management*: Vol. 14: Iss. 3, Article 4.  
Available at: <http://scholarworks.lib.csusb.edu/jitim/vol14/iss3/4>

This Article is brought to you for free and open access by CSUSB ScholarWorks. It has been accepted for inclusion in Journal of International Technology and Information Management by an authorized administrator of CSUSB ScholarWorks. For more information, please contact [scholarworks@csusb.edu](mailto:scholarworks@csusb.edu).

# UML Activity Diagram Semantics and Automated GUI Prototyping

**Nancy Winniford Ashley**  
**Washington State University, Richland, WA**

**Timothy E. Meehan**  
**Nuvotec, Inc., Richland, WA**

**Norman Carr**  
**Nuvotec, Inc., Richland, WA**

## ABSTRACT

*Extended Activity Semantics (XAS) is a notation which can be used with Unified Modeling Language (UML) activity diagrams to specify user interactions with a system and to automatically generate a prototype of the graphical user interface (GUI) that would be used in these interactions. XAS has been incorporated in a CASE tool, Guibot, which has been developed as a plug-in for Rational Rose, a leading UML tool. The notation and tool address a specific gap in UML – the inability to model user interaction.*

## INTRODUCTION

While we are not likely to find a silver bullet to solve all of the problems of the system development lifecycle (Fedorowicz, 2004), a tool which speeds prototype development and adds precision to design specifications has promise for addressing problems in the requirements elicitation phase of the lifecycle. Extended Activity Semantics (XAS) is a notation which can be used with Unified Modeling Language (UML) activity diagrams to specify interactions with a system in a succinct style. XAS can be included in a modeling tool to automatically generate a graphical user interface (GUI) prototype. The notation and tool address a specific gap in UML – the inability to model user interaction.

## LITERATURE REVIEW OF CURRENT APPROACHES USED IN REQUIREMENTS ELICITATION

Most phased, iterative system development methodologies recognize that involving users as much as possible, preferably continually, throughout the system development process is key to successful elicitation of requirements (Hoffer, George, & Valacich, 2005). It is the requirements elicitation process with which this paper is particularly concerned.

Two main approaches are used to represent the elicited requirements and communicate them back to the users – modeling and prototyping. Prototyping user interfaces has been recognized as particularly effective in improving communication between the user and developer, and results in an improved final product (Phillips & Kemp, 2002; Hoffer et al. 2005). Seffah and Andreevskaia (2003) state that creating effective prototypes to model screen layouts and user interaction is a skill needed by software engineers to effectively conduct user-centered design, which aims at increasing usability. Effective as prototypes are for communicating, however, they are not a substitute for precise modeling.

Object-oriented techniques are recognized as powerful, and the Unified Modeling Language (UML) is emerging as an apparent modeling standard, but thus far it has not solved all modeling problems. Rational Rose is a UML-based modeling tool which can represent many types of detailed system requirements and behaviors.

Several problems are inherent in software modeling when addressing the end-user concerns. First is the

problem of level of detail. Some diagrams, like sequence diagrams, can capture an enormous amount of detail, but are difficult for users to understand. If users cannot understand the diagram, they cannot confirm that requirements have been correctly understood by software developers (Wilcox, 2003). Other simpler diagramming techniques, such as use case and activity diagrams, may be easier for the user to understand, but do not capture sufficient detail to actually build a correctly functioning system (Phillips, Kemp, & Keck, 2001). Worse yet, modeling the user interface, which is the part of the system the user best understands and can best give feedback on, is not directly supported in UML (Scogings & Phillips, 2001, Meehan & Carr, 2005).

Prototyping is not without problems either. The most obvious problem is the time and expense that is required to build a functional prototype (Mannio & Nikula, 2001; Phillips & Kemp, 2002). Once the time and expense has been expended to create a functional prototype, there is a tendency on the part of both the developer and the user to commit to the technology and interface represented by the prototype, rather than exploring more options (Phillips & Kemp). Beyond these practical difficulties, which might be overcome with enough time and money, is the fact that prototypes generally provide fairly static screen shots, and simply do not give an easily understood dynamic view of the workflow involved in a process (Phillips and Kemp, Meehan & Carr, 2005). Because the prototypes do not provide clear understanding of the processes being modeled, users can come away from the design process with inaccurate expectations about how the system will behave in the workflow (Mannio & Nikula, Phillips & Kemp, Meehan & Carr), and developers can wind up building a system based on incorrect requirements.

Even when developers utilize both prototyping and modeling tools, they can, and frequently do, still have problems effectively capturing correct system specifications and communicating them to their users. The problem is that, although designers may build prototypes and create models, there is no built-in way in UML to link the prototypes to the models in a way that accurately describes the processes and workflow in sufficient detail. What is needed is a link between user-intuitive prototypes and models capable of capturing adequate detail. Phillips, Kemp and Kek identified the need to model user tasks “within the context of the visible interface” (2001, p. 49). In other words, while it is important to model tasks and behaviors and their attendant detail, there needs to be available a visual prototype of the user interface that users can refer to ensure understanding.

### **NEEDED: A LINK BETWEEN A STRUCTURED MODEL AND A GRAPHICAL PROTOTYPE**

While sequence diagrams are capable of capturing significant amounts of detail, Wilcox (2003) identified three structured diagramming techniques – packages, use cases, and activity diagrams – that were better understood by users and developers than sequence diagrams. Use cases and packages do not address specifically the requirements elicitation process of the system development lifecycle, which is the concern of this paper. The purpose of use case diagrams is to identify high level goals for the system – an important task that should occur before detailed requirements elicitation in the systems development lifecycle. Packages are simply a tool for visually grouping related system elements (as folders are used in an operating system), and do not specifically address requirements elicitation. Activity diagrams, however, can represent the tasks that users are describing in sufficient detail that programmers can work directly from the activity diagram in the subsequent implementation processes of the system development lifecycle. Therefore, by linking these detail-laden activity diagrams to visual prototypes we can model tasks with sufficient detail to support later implementation phases while at the same time having a visual prototype available for communicating with users during the requirements elicitation phase.

There are various methods of prototyping, and different kinds of prototypes. For example, some methodologies recommend evolutionary prototyping, in which the prototype is refined until it becomes the target system. Those with concerns about locking in too early on development platforms or interface concepts will avoid the evolutionary approach in favor of the throwaway prototype. Asur and Hufnagel (1993) suggest that nonexecutable and visual prototypes are effective, cheap, and fast approaches that can be used effectively to clarify system behaviors. Non-executable prototypes are, of course, eminently easy to throw away, since they do not actually work at all.

## RESEARCH CONTRIBUTION: DEVELOPMENT OF A NOTATION FOR USER INTERACTION, AND A CASE TOOL

The authors developed Extended Activity Semantics (XAS) for use in activity diagrams, and implemented it in a CASE tool which makes the link between the activity diagram and the visual prototype. The authors found in their custom software development practice that using XAS allowed them to capture greater detail about user interactions as they used it to model business tasks. User interactions include, for example: enter this, read that, select next, submit all. Each element of XAS notation can be directly mapped to one or more GUI components, such as list boxes, text boxes, and buttons. The authors created a CASE tool which incorporates XAS and the mapping of the XAS notation to GUI components. By using the CASE tool, a prototype of the user interface composed of the mapped GUI components, is constructed simultaneously as detailed information about the interface is captured with the XAS notation in the activity diagram. Using the CASE tool, designers and users can switch between screens, or have both screens open at once, watching how, as detailed information about user behavior is captured in activity diagrams, a visual prototype of the user interface is automatically assembled on another screen.

### Relevant User Behaviors and their Representation in XAS

As the authors worked with users in many different development projects, they found that user interactions with interfaces could always be described with a set of four irreducible actions — input, review, select, and command. In creating XAS, they represented these four actions as inputters, outputters, selectors, and action invokers. User actions such as drag-and-drop are composite actions of these irreducible actions, where drag-and-drop would be select-and-input.

The authors also recognized that to provide adequate detail to support subsequent implementation by programmers, more information about these four basic interactions needed to be captured and represented in XAS. Therefore notation was included to describe multiplicity, labels, formatting, and conditions, which are described below.

Multiplicity can indicate a couple of different things, depending on the user behavior. For any user action, multiplicity can denote whether the action is required or optional. If the user is making a selection, multiplicity denotes whether one or many selections can be made.

The Label notation is another feature of XAS which, like multiplicity, provides different functionality depending on the user behavior. Users can better understand the GUI components on the prototype if they are identified with their business language. The label notation in XAS creates labels such as “Customer Name.” for a field where the data will be inputted. Another use of the label notation is to identify a data type, for instance, identifying the data inputted in the Customer Name field as “text”. A third use of the label notation is to identify the action that is taken on a command behavior, for instance, that the command is “click to save record.”

The formatting filter allows the specification of the presentation of the information, such as the format of a date as mm/dd/yyyy.

Sometimes a user action is dependent upon a guard condition, and these conditions can be represented in XAS. Two examples are “date entry must be earlier than today,” and “password must be longer than some number of characters.” These examples represent a general principle in XAS, which is to allow actions and constraints to be described in the business language of the end user, which we have found most effective during requirements elicitation.

To summarize, we added the following XAS notation to activity diagrams to translate typical business process behavior into user-system interactions which map to GUI components in the prototype.

- Inputter:  $\gg$ , where the user provides information to the system.
- Outputter:  $\ll$ , where the system provides information to users.
- Selector:  $\nabla$  or  $\mathbf{V}$ , where users select from a predetermined choice of output.
- Action Invoker:  $!$ , where users signal the system to provide an action or continuance.

- Multiplicity: **m..n** defines the multiplicity of each user interaction.
- Label : **DataType**, which specifies the plain-language label of the data and the data type, or the command for the action invoker.
- Mask/Filter: |, where the data format is represented.
- Condition: [ ], where any conditions applied to the interaction are specified.

The complete grammars for the four user behaviors are presented below:

Inputters, outputters, and selectors:

>>, <<, V m..n label : datatype | mask [conditions]

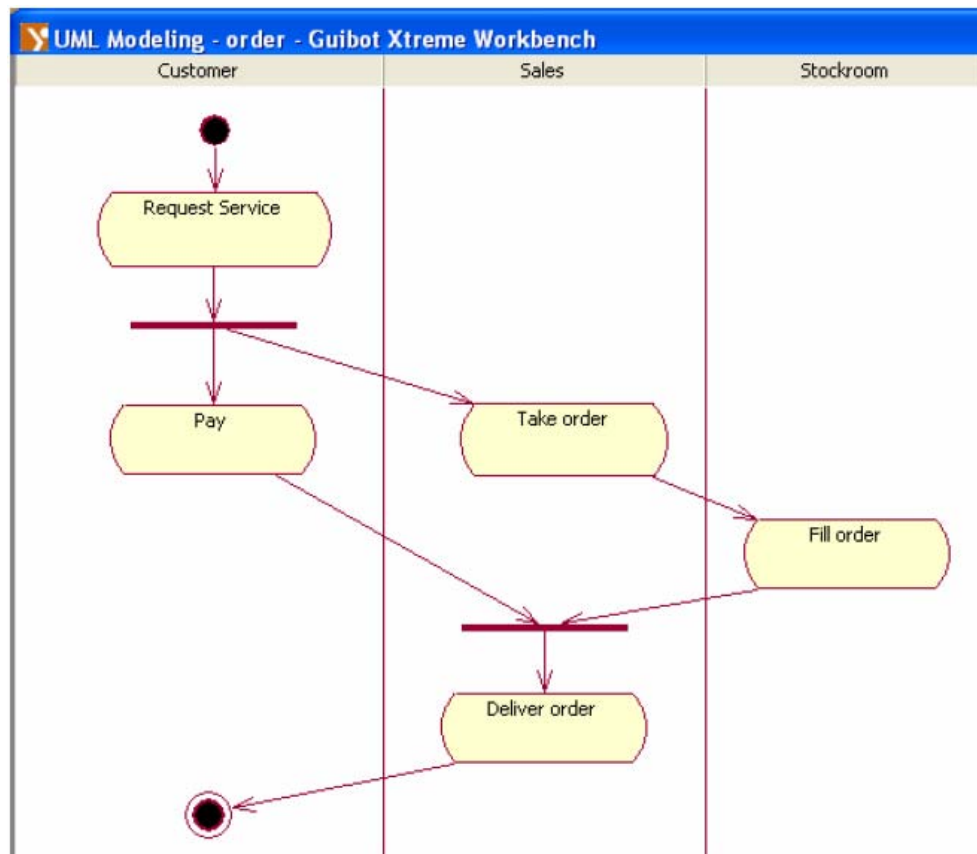
Action invokers:

! command [conditions]

### Examples Showing the Application of XAS

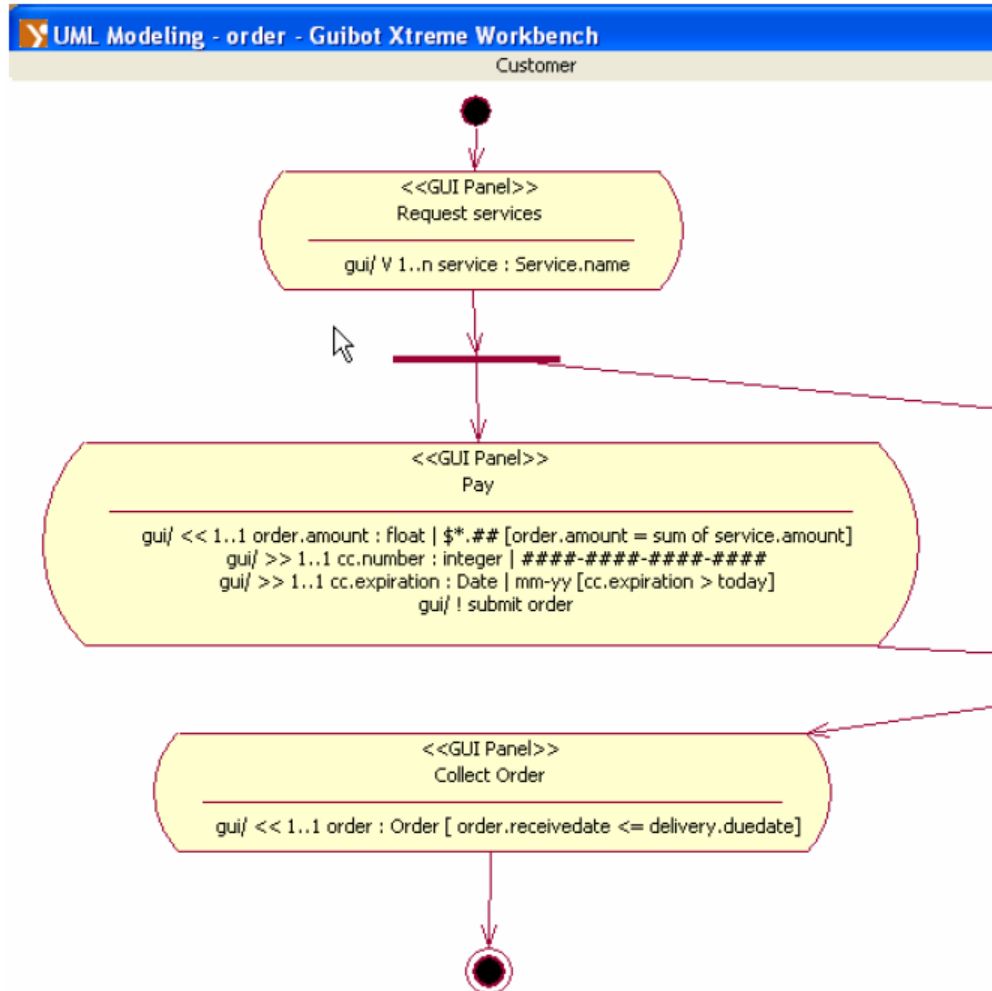
Figure 1, a typical activity diagram presented in the UML 1.5 specification, illustrates customer ordering services from the sales department and a stockroom filling the order. While this example is a satisfactory description of the general workflow, these action states require more detail to describe the process for both end users and software developers when it comes to human/computer interface issues.

**Figure 1: Typical activity diagram from UML 1.5 specification.**



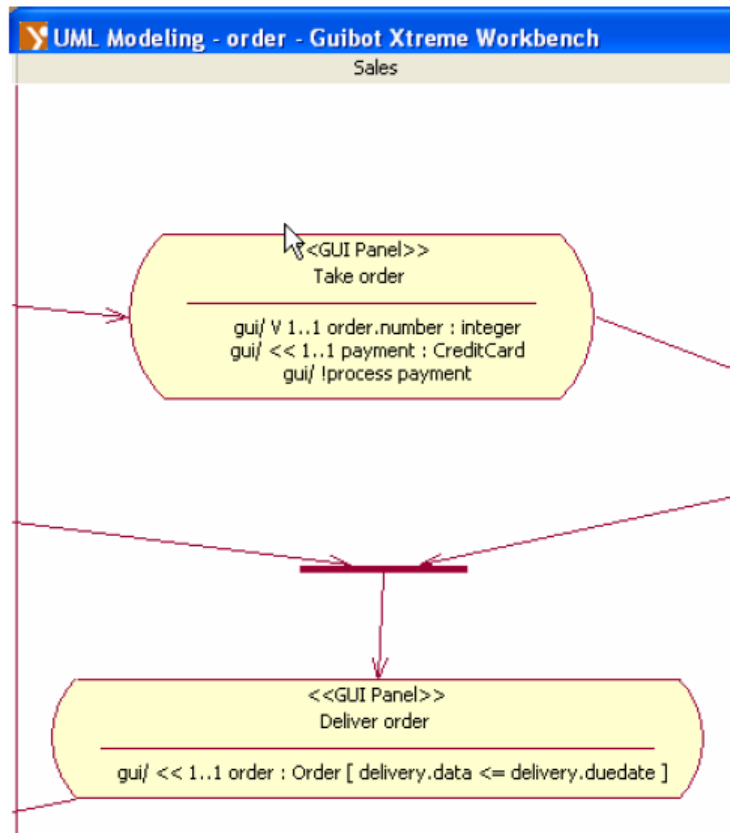
By adding XAS to each action state that requires user interaction in Figure 1, we can detail the interactions required and any requirements surrounding these user interactions that are necessary to complete the task. Figure 2 is a more detailed activity diagram employing XAS and more comprehensively describing user activity.

Figure 2: Addition of XAS to the action states in Figure 1 for the customer swimlane.



In Figure 2, in the first action state the customer selects at least one, or possibly several (1.. n) services. In the next action state, we see that the system outputs the order amount, formatted in this case as \$\*.##. The order amount is specified with the condition that the order amount must equal the sum of the amounts of each individual service. The credit card number and expiration date are also inputted, with appropriate formatting, followed by the command to submit the order. In the third action state, the ordered goods are delivered by the system and collected by the customer. A condition has been included so that the order indicates that the receivable (or collection) date for the order is not earlier than the delivery due date. This example illustrates how activity diagrams and XAS can be used to model hardware interfaces and physical processes of the total system in connection with the software user interfaces.

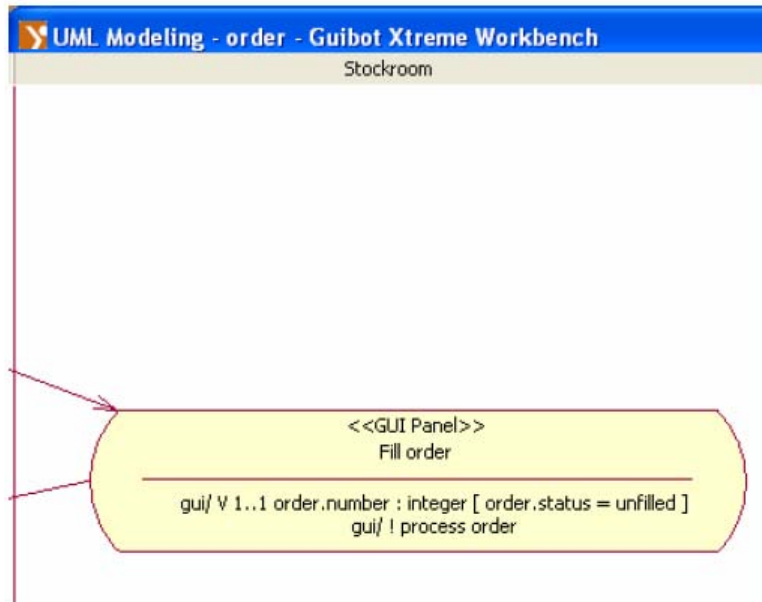
Figure 3: Addition of XAS to the action state in Figure 1 for the sales swimlane.



In Figure 3, the sales department selects the order, reviews the payment submitted, and processes the payment. When the order is filled by the stockroom, the sales department delivers the order to the customer with the specified condition that the delivery date is less than or equal to the delivery due date.

In Figure 4, the stockroom selects an order with the condition it has not been filled, and then processes the order in order to fill it.

Figure 4: Addition of XAS to the action state in Figure 1 for the stockroom swimlane.



### Mapping XAS to the GUI

As stated earlier, to automatically generate GUI prototypes, we have developed a mapping between the activity diagram elements and the notation elements of Extended Activity Semantics; see Table 1. Swimlanes represent the view for the actor (identified in a previously created use-case diagram) and are represented by a GUI form. The action state groups the user interaction and is represented by GUI groupboxes or panels. Inputters typically are text boxes, text areas, and grids. Outputters typically are labels, grids, image boxes, and read-only text areas. Single selectors typically are comboboxes, single selection list boxes, radio buttons, or treeviews. Multiselectors are typically multiselection list boxes, grids, checkboxes, and list views. Action invokers are typically command buttons, hyperlinks, menu items, and image buttons.

Table 1: XAS to GUI component mapping.

Activity Diagram Element/Notation	GUI Component
Swimlane	Form
Action State	Groupbox, Panel
Inputters >>	Input box, text area, grid
Outputters <<	Label, grid, image box, text content.
Selectors 0..1 (single selection)	Combobox, list box, radio button, tree view
Selectors 0..n (multiple selection)	List box, grid, checkbox, list view
! command	Command button, hyperlink, menu, image button

Guibot is the CASE tool created by the authors incorporating XAS and directly mapping the activity diagram elements and the XAS to GUI components. As the activity diagrams in Figures 2, 3 and 4 are created, Guibot automatically produces the GUI prototypes for the customer, sales, and stockroom swimlanes as shown in Figures 5, 6, and 7. The resulting prototype allows immediate understanding between users and analysts, and defines the expectations for the coder.



Figure 5: GUI prototype produced by XAS notation in the customer swimlane in Figure 2.

UML Modeling - order - Guibot Xtreme Workbench

Customer Sales Stockroom

Request services

service : Service.name

Line 1

Line 2

Line 3

Pay

order.amount : float | \$\*.## [order.amount = sum of service.amount]

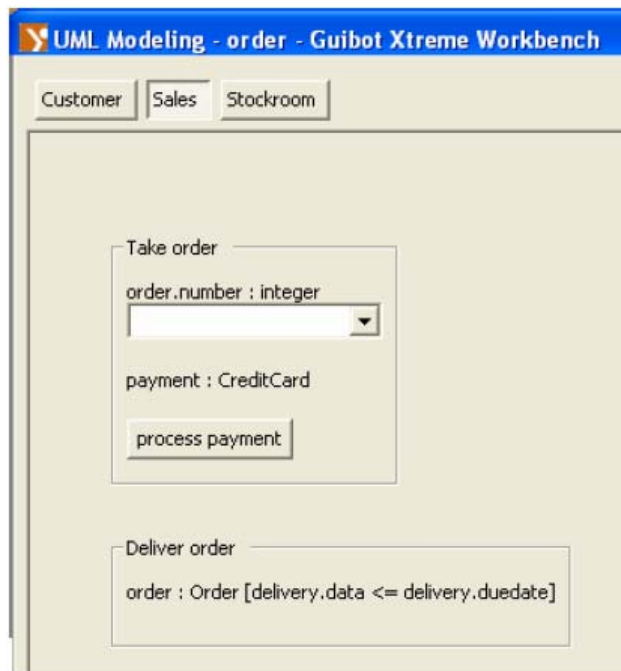
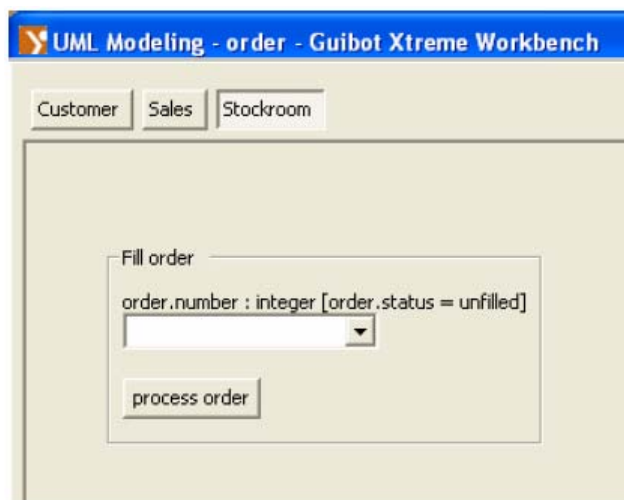
cc.number : integer | ####-####-####-####

cc.expiration : Date | mm-yy [cc.expiration > today]

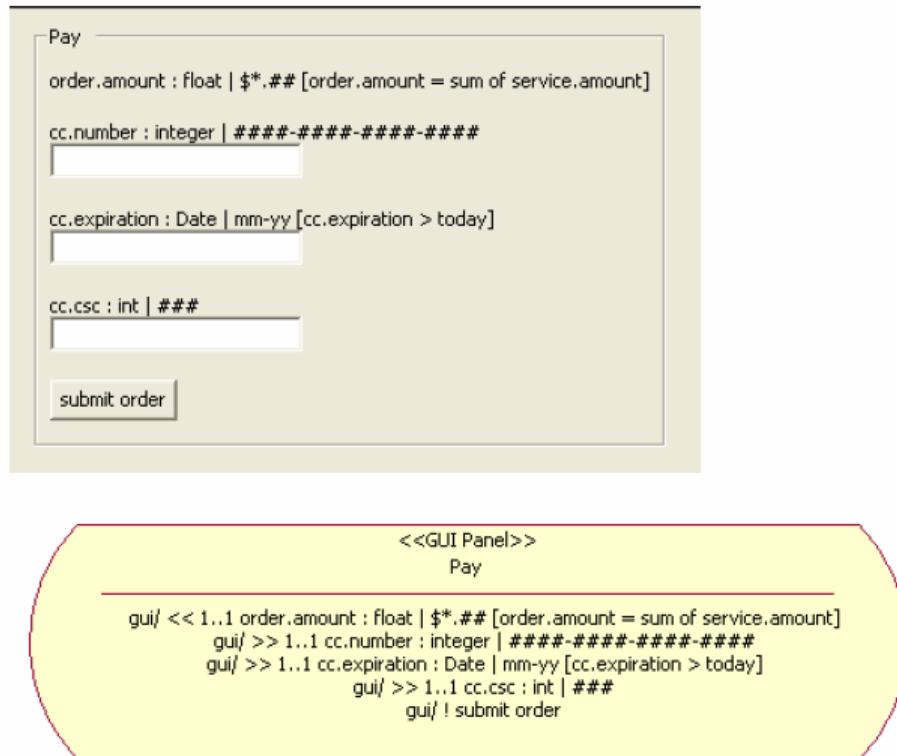
submit order

Collect Order

order : Order [order.receive date <= delivery.due date]

**Figure 6: GUI prototype produced by the XAS notation in the sales swimlane in Figure 3.****Figure 7: GUI prototype produced by the XAS notation in the stockroom swimlane in Figure 4.**

An action state can also be reverse-engineered by working directly with the prototype. If you add an input box to the Pay panel in the Customer user interface in Figure 5 for security code (cc.csc) entry for credit cards, the action state is updated with the new inputter information via the mapping between notational elements and the GUI components; see Figure 8.

**Figure 8: Reverse engineering of GUI component to XAS notation for an action state.**

In our practice we have found that use-case diagrams and activity diagrams with XAS are frequently the only diagramming tools that we use when eliciting software and system requirements from end-users.

### Mapping Extended Activity Diagrams to Object-Oriented Class Diagrams

Since XAS captures the information for the label and data type, which is typically modeled in class diagrams, the extended activity diagrams and the GUI components now have a direct link to the attributes within a class. If a class does not exist in the class diagram to provide the necessary entity and datatype, then a new class is entered into the class diagram to fulfill these user requirements. Furthermore, the extended notation illustrates the origin of object flow in an activity diagram, which allows the GUI component, object flow within an activity diagram and class diagram to be linked with their requisite behavior specified.

XAS is incorporated in Guibot, which is a plug-in for Rational Rose, the leading UML modeling tool. While there are as yet relatively few users of the tool Guibot, there is a wide need for the abilities found in XAS to enhance communication between end-users and analysts. We know of no comparable tools or notations that support the collection of precise requirements in parallel with the generation of a prototype interface.

### Call for Further Research

Further research is needed on XAS. We would like to see implementations of XAS in other UML tools, and refinement of Guibot. Especially needed are controlled studies to determine its effectiveness in capturing requirements and communicating with users, compared to other approaches.

## CONCLUSION

The developers of XAS and Guibot have seen in their software development practice that XAS extensions maximize the benefits that activity diagrams offer. In our practice we have seen that rapid prototyping gathers faster buy-in from the customer and faster elaboration of their business needs. Less ambiguous software requirements are generated earlier, obviating the need for onerous or extended iterations. This method yields greater end-user involvement in the design and specification of user activity, provides better user satisfaction, and results in a more accurate and detailed definition of end-user requirements early in the project lifecycle. The extended activity diagrams are implementation agnostic, allowing technologists to determine the best implementation path, and avoiding the excessively early buy-in to a development platform that has been seen to be a problem in evolutionary prototyping. By addressing the need for a link between a diagram containing adequate detail (activity diagrams with XAS notation) and a complementary visual prototype, it is possible to provide accurate vision to stakeholders of the proposed design.

## REFERENCES

- Asur, S. and Hufnagel, S. (1993). Taxonomy of rapid-prototyping methods and tools, *Proceedings from the IEEE Fourth International Workshop on Rapid System Prototyping*, June 1993, 42-5.
- Fedorowicz, J., J. L. Gogan, & A. W. Ray (2004). The Ecology of Interorganizational Information Sharing. *Journal of International Technology and Information Management (formerly Journal of International Information Management)* 13(2), 73-86
- Hoffer, J.A., George, J.F., and Valacich, J.S. (2005). *Modern systems analysis and design*. (4<sup>th</sup> ed.) Upper Saddle River, N.J.: Pearson Prentice Hall.
- Mannio, M. and U. Nikula (2001). Requirements elicitation using a combination of prototypes and scenarios, *IV Workshop on Requirements Engineering, Buenos Aires, Argentina, Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Argentina*, 2001, 283-296.
- Meehan, T. E. and Carr, N. (2005). Extending UML, *Dr. Dobb's Journal*, February 2005, 56-60.
- Phillips, C. and Kemp, E. (2002). In support of user interface design in the rational unified process, *Australian Computer Science Communications, Third Australasian Conference on User interfaces, Melbourne, Victoria, Australia*, January 2002, 21-27.
- Phillips, C.H.E., Kemp, E.A. and Kek, S.M. (2001). Extending UML use case modelling to support graphical user interface design, *Proceedings of ASWEC 2001, IEEE, Canberra, Australia*, 26-28 August 2001, 48-57.
- Scogings, C. and Phillips, C. (2001). A method for the early stages of interactive system design using UML and Lean Cuisine+, *Australian Computer Science Communications, Proceedings of the 2nd Australasian Conference on User Interface*, January 2001, 69-76.
- Seffah, A. and Andreevskaia, A. (2003) Empowering software engineers in human-centered design, *Proceedings of the 25th International Conference on Software Engineering*, May 2003, Portland, OR, 653.
- Wieggers, K. (1995). In search of excellent requirements, *Journal of the Quality Assurance Institute*, 9(1).
- Wilcox, P. A. (2003). Effective communication of scenarios of usage, *A HWISE (Heriot Watt University Institute of Software Engineering) Technical Report, OPHELIA (Open Platform and Methodologies for Development Tools Integration in a Distributed Environment) Project*, Retrieved December 12, 2004 from <http://www.macs.hw.ac.uk:8080/techreps/index.html>

