

Wisdom – A UML Based Architecture for Interactive Systems

Nuno Jardim Nunes¹ and João Falcão e Cunha²

¹ Universidade da Madeira, Unidade de Ciências da Computação, Dep. de Matemática,
9000 Funchal, Portugal
n.jn@math.uma.pt

² Universidade do Porto, GEIN, Faculdade de Engenharia,
4099 Porto CODEX, Portugal
jfcunha@fe.up.pt

Abstract. The UML is recognized to be the dominant diagrammatic modeling language in the software industry. However, its support for building interactive systems is still acknowledged to be insufficient. In this paper we discuss and identify the major problems using the UML framework for interactive system development, specifically, in what concerns the architectural issues. Here we present a conceptual architectural model that expands the analysis framework of the Unified Process and the UML profile for software development processes. Our proposal leverages on user-interface domain knowledge, fostering co-evolutionary development of interactive systems and enabling artifact change between software engineering and human-computer interaction, under the common notation and semantics of the UML.

1 Introduction

Until recently both research and practice didn't focus on the overall organization and coordination of the different processes, models, artifacts and presentation aspects of interactive systems. In fact, it is even difficult to find a clear definition of user interface architecture in the literature.

Artim describes a user interface architecture as "an approach to development processes, organization and artifacts that enables both good UI practices and better coordination with the rest of development activities" [1]. This highly conceptual description focuses mainly the process of building an interactive system and the need for collaboration and coordination between HCI and SE practice. Paternò gives a more traditional (in the software engineering sense) definition "the architectural model of an interactive application is a description of what the basic components of its implementation are, and how they are connected in order to support the required functionality and interactions with the user" [2]. However, this definition, still lacks some key aspects of software architectures, like reuse and pattern composition. Kovacevic adds reuse concerns and some key aspects of model based approaches observing that such an architecture

should "maximize leverage of UI domain knowledge and reuse (...) providing design assistance (evaluation, exploration) and run time services (e.g., UI management and context-sensitive help)" [3].

In this paper we claim that an architecture for interactive systems involves the description of the elements from which those systems are built, the overall structure and organizations of the user interface, patterns that guide their composition and how they are connected in order to support the interactions with the users in their tasks.

In the following sections we discuss different contributions for a clear definition of interactive system architecture and present a new UML based architecture. In section 2 we discuss the conceptual and implementation models devised to support interactive system development and address the major concerns in this architectural evolution process. In section 3 we describe the Unified Process [4] analysis framework, discuss how it relates to the user interface conceptual architectures and describe efforts for bridging the gap between both worlds. In section 4 we present a new UML based architecture for interactive systems and provide examples of its expressiveness using the appropriate UML extensions. Finally section 5 presents our conclusions and further work.

2 Software Architecture Models for Interactive Systems

Work on user interface architecture initially started with concerns about conceptual architectural models for user-interface objects (or entities). Examples of conceptual architectural models, depicted in Fig. 1, are MVC [5], PAC [6] and the Lisboa Model [7]. Those models introduced the concepts of abstraction and concurrency to the user interface domain, through a set of collaboration agents (MVC and PAC) or objects (Lisboa). For a discussion and classification framework of conceptual architectural models refer to [8].

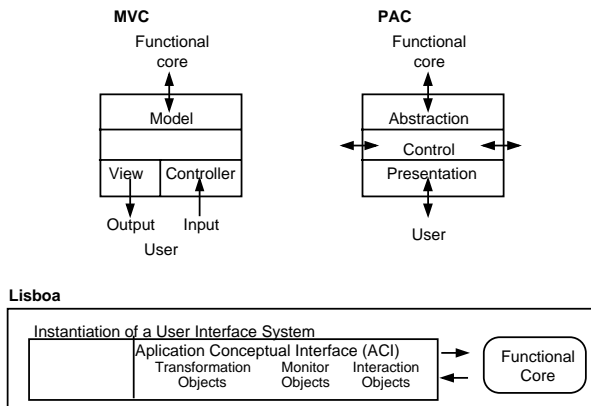


Fig. 1. Conceptual Architectural Models for Interactive Systems

In the late 1980s and early 1990s several approaches of implementation architectural models for interactive systems emerged. Unlike conceptual models, implementation models organize the interactive system as a set of software components and aim at practical software engineering considerations like reusability, maintainability, and performance.

The Seeheim [9] model proposes a simple three-layer model (application, dialogue and presentation) of an interactive system, roughly coupling the semantic, syntactic and lexical functionality's of the user interface. The application component models the domain-specific components, the dialogue component defines the structure of the dialogue between the user and the application and, finally, the presentation component is responsible for the external to internal mapping of basic symbols. The Seeheim model is considered a correct and useful model for specification of interactive systems. However, its support for distribution, concurrency, resource management and performance is recognized to be insufficient [8].

The Arch model [10] is a revision of the Seeheim model aimed at providing a framework for understanding of the engineering tradeoffs, specifically, in what concerns evaluating candidate run-time architectures. The Arch model proposes a five-layer approach balanced around the dialogue component. The components of the Arch model are:

- The interaction toolkit component implements the physical interaction with the end-user;
- The presentation component provides a set of toolkit independent objects;
- The dialogue component is responsible task-level sequencing, providing multiple interaction space consistency, and mapping the between domain specific and user interface specific formalisms;
- The domain adapter component implements domain related tasks required but not available in the domain specific component;
- The domain specific component controls, manipulates and retrieves domain data and performs other domain related functions.

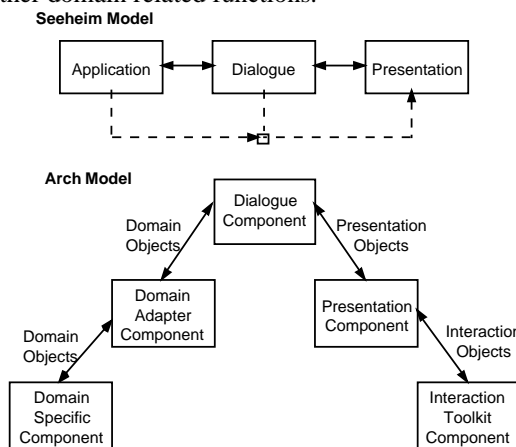


Fig. 2. Reference implementation models for interactive systems: The Seeheim and Arch models

The Arch model provides developers with guidance to tackle the difficult engineering compromises that affect the development process of interactive systems and the quality of the end product. On the one hand the user and the functional core play a symmetric role driving the dialogue component, hence, at high levels of abstraction there is no *a priori* imposition on the control of the interaction. On the other hand, both the domain adapter and interaction toolkit components serve as buffers for the functional core and the user, therefore, isolating and absorbing the effects of change in its direct neighbors [8].

3 The Unified Process Analysis Framework

3.1. The Original OOSE Analysis Framework

In their initial proposal for the object-oriented software engineering (OOSE) approach, Jacobson and colleagues introduced the concept of use-cases and defined the entity, interface and control analysis framework [11].

The information space for this analysis framework defines three dimensions (depicted in Fig. 3):

- The information dimension specifies the information held in the system in both short and long term – the state of the system;
- The behavior dimension specifies the behavior the system will adopt – when and how the system’s state changes;
- The interface dimension specifies the details for presenting the system to the outside world.

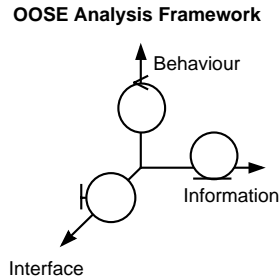


Fig. 3. The dimension space of the OOSE analysis framework

The reason behind this partitioning of analysis classes into information, behavior and interface is to promote a structure more adaptable to changes. Assuming that all systems change, stability will occur in the sense that changes affect (preferably) only one object in the system, i.e., they are local [4]. Therefore, the OOSE analysis framework aims at concentrating changes to the interface and related functionality in interface objects; changes to (passive) information and related functionality in entity objects; and changes to complex functionality (e.g., involving multiple objects) in control objects.

This approach is conceptually similar, although at a different granularity level, to the PAC model presented in the previous section. In fact, the PAC distinction between presentation, abstraction (application) and control relate, conceptually, to the interface, entity and control objects.

3.2. The Unified Process Analysis Framework

With the advent of the UML, unifying the major object-oriented methods and notations, the OOSE framework was adopted by the Rational Approach leading to the Rational Unified Process (RUP) [12] and later to what is known as the Unified Process [4]. The influence of this approach can also be found in the UML standard, in the form of an extension profile for software development processes [13]. In this evolutionary process subtle conceptual and notational changes occurred. The original interface dimension and the interface object are known in the Unified Process as, respectively, presentation dimensions and boundary class stereotype.

The authors of the Unified Process claim that this process is use-case driven, architecture centric and iterative and incremental [4]. Despite the importance of the iterative and incremental nature of the process for adequate interactive system development – as suggested in the ISO standard for user-centered design – it is not our aim here to discuss process related issues. For a discussion of process issues refer to [14] [15].

The same concerns apply, at a different level, to the use-case driven aspect of the unified process. However, since use-cases drive the architecture, and the architecture drives the selection of use-cases, we believe that some of the architectural problems lie within the intrinsic definition of what is a user (and a use-case) in the unified approach. Jacobson and colleagues claim that "the term user refers not only to human users but to other systems (...) someone or something that interacts with the system being developed" [4]. It is our belief that such definition is, in part, responsible for the system-centric nature of use-cases and, as a consequence, conditions a system-centric architecture. In a proposal for essential use-cases Constantine also argued about the imprecise definition and system-centric nature of use-cases [16]. The essential use-case extension enabled the connection between the structure of use and the structure of the user interface, providing a way to connect the design of the user interface back to the essential purpose of the system and the work it supports. However, such concerns about the system-centric nature of use-cases were not considered in the unified process.

The architecture-centric nature of the unified process is responsible for the focus on the early development and base-lining of a software architecture, used as a primary artifact for conceptualizing, constructing, managing and evolving the system under development [4]. Such architecture is responsible for the form that, together with the function captured in the use-case model, shapes the evolution of the software system. The architecture is expressed in the unified process in terms of different UML artifacts (subsystems, nodes, active classes, interfaces, etc.) and grouped in different views.

Although the Unified Process and the UML notation provide a multitude of architectural representations, in the remainder of this paper we concentrate in the analysis framework defined in the analysis model of the Unified Process. We consider the analysis framework an architectural description because the elements that establish the analysis model (boundary, entity and control), and the corresponding relationships amongst them, organize the system defining how the different elements (classes) participate in the realization of the functional requirements captured in the use-case model. We refer to the Unified Process Analysis framework as the Unified Process Analysis Architecture (UPA architecture for short), meaning the overall organization of analysis classes supporting the conceptualization, construction, management and evolution of the system.

3.3 Bridging the Gap

Between 1997 and 1999, several workshops held at major HCI and OO conferences (CHI and ECOOP), discussed the role of object models and task/process analysis in user interface design [17] [18] [15]. All this work, emerging both from industry and academia, stressed the importance, and the opportunity, of using object-technology to bridge the gap between software engineering and human-computer interaction practice. Although the workshops addressed many different issues, a general framework devised at the CHI'97 workshop [17] was consistently used to illustrate an object-oriented conceptual architecture for interactive systems. Fig. 4 illustrates the Wisdom'99 [15] revised version of the original CHI'97 framework.

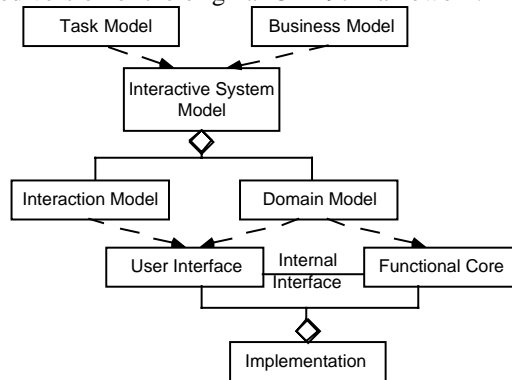


Fig. 4. The Wisdom'99 version of the CHI'97 framework

This conceptual model architecture clearly (logically, not necessarily physically) separates the functional core from the user interface. Note that, while the user interface depends on both the domain and interaction models, the functional core only depends on the domain model. This conceptual model also supports the separation of the presentation (in the user interface) from the conceptual specification of the dialogue (the interaction model) [15] [3].

4 The Wisdom Architecture

In the previous sections we discussed different approaches and contributions to define adequate interactive system architectures. We also discussed the existing Unified Process analysis architecture and argued about its system-centric nature. In this section we start from the unified process analysis architecture (UPA architecture) and expand this approach to accommodate the dialogue and presentation components, present, in the MVC and PAC conceptual models.

4.1 The Dimensions of the Wisdom Architecture

In the previous section we discussed the system-centric nature of the use-case model, and its impact in the corresponding analysis architecture model. Our proposal for a new architecture for interactive systems introduces a user-centered perspective at the analysis level. Despite all the requirements for interactive system architecture, while devising the new architecture model, we complied with the following criteria:

- the architecture model should build on user interface design knowledge capturing the essence of existing successful and tested architectural models;
- the architecture model should seamlessly integrate the existing unified analysis model, while leveraging cooperation, artifact change and ensuring traceability between human-computer interaction and software engineering models;
- the architecture model should foster separation of concerns between internal functionality and the user interface, not only in its intrinsic structure, but also enabling the co-evolution of the internal and interface architectures;
- the architecture model should conform to the UML standard both at the semantical and notational levels, i.e., the artifacts required to express the architecture should be consistent with the UML and its built-in mechanism;

The information space for the Wisdom UML based architecture is depicted in Fig. 5. The new information space, contrasts the information space of the OOSE and UPA (see Fig. 3), introducing two new dimensions for the dialogue and presentation components. Additionally the UPA presentation dimension is restricted to non-human interface. Note that the information dimension is shared between the two information spaces, leading to a total of five dimensions if we consider both information spaces as a whole.

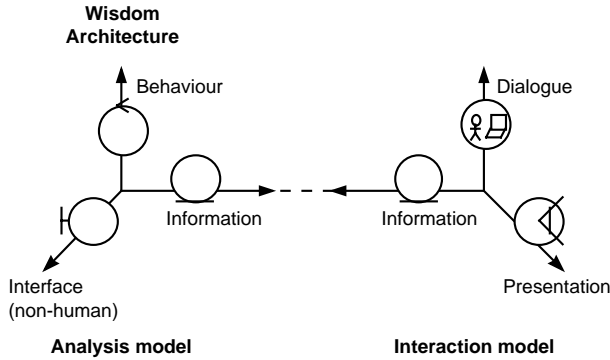


Fig. 5. The Arch model and the Wisdom architecture

The Wisdom architecture proposes a new UML interaction model to accommodate the two new dimensions, plus the shared information dimension. Therefore, the interaction model encompasses the information, dialogue and presentation dimensions, clearly mapping the conceptual architectural models for interactive systems. Accordingly, the analysis model accommodates the existing UPA architecture dimensions, also including the shared information dimensions. Note that the presentation dimension in the UPA architecture is reduced to capture the interface (not the presentation) to external systems. This way we are able to create two information spaces, sharing a common dimension (information) that ties the internal and interface architectures, leveraging the separation of concerns. Moreover, the two information spaces accommodate the domain knowledge of both the OOSE and HCI (or user interface) communities.

The Wisdom architecture - like the MVC, PAC and UPA analysis architectures - is a conceptual architecture. Therefore it should not take into consideration design or implementation criteria, like the Seeheim and Arch models do. However, the Wisdom architecture is at a higher granularity level of the MVC and PAC models, hence, it will eventually suffer subsequent reification at design and implementation. Therefore the Wisdom architecture should support such reification, maintaining qualities like robustness, reuse and location of change; while leveraging the mediating nature of the domain adapter and interaction toolkit components of the Arch model (Fig. 2). This process is typically achieved through precise allocation of information objects (entity classes) to domain adapter and domain specific components at design or implementation time. Such allocation enables semantic enhancement (dividing or joining objects in the domain adapter component) and semantic delegation (enhancing performance by preventing long chains of data transfer to objects in the domain specific component). The same applies to the interaction toolkit component, at this level with presentation objects (interaction space classes).

4.2 Elements of the Analysis Model

In the Wisdom architecture the analysis model concerns the internal architecture of the system. This model structures the functional requirements in terms of analysis classes, postponing the handling of non-functional requirements to subsequent design and implementation models.

The elements of the analysis model are analysis classes, standardized in the UML as class stereotypes [13]. The three stereotypes are [4]:

- Boundary class – the boundary class is used, in the Wisdom architecture, to model interaction between the system and external systems (non-human actors). The interaction involves receiving (not presenting) information to and from external systems. Boundary classes clarify and isolate requirements in the system’s boundaries, thus isolating change in the communication interface (not human-interface). Boundary classes often represent external systems, for example, communication interfaces, sensors, actuators, printer interfaces, APIs, etc.
- Control class – the control class represents coordination, sequencing, transactions and control of other objects. Control classes often encapsulate complex derivations and calculations (such as business logic) that cannot be related to specific entity classes. Thereby, control classes isolate changes to control, sequencing, transactions and business logic that involve several other objects.
- Entity class – the entity class is used to model perdurable information (often persistent). Entity classes structure domain (or business) classes and associate behavior, often, representing a logical data structure. As a result, entity classes reflect the information in a way that benefits developers when designing and implementing the system (including support for persistence). Entity objects isolate changes to the information they represent.

4.3 Elements of the Interaction Model

The new interaction model, proposed in the Wisdom architecture, concerns the overall organization of the user interface in an interactive system. This model structures the interaction with users (human actors) in terms of interaction classes, postponing the handling of user interface styles, technologies and other constraints to subsequent design and implementation models.

The elements of the interaction model (Fig. 6) are interaction classes, defined as stereotypes of UML class constructs. The two stereotypes proposed in the Wisdom architecture are:

- Interaction space class¹ – the interaction space class is used to model interaction between the system and the users (human-actors). A interaction space class represents the space within the user interface of a system where the user interacts with all the functions, containers, and information needed for carrying out some par-

¹ In previous versions of the Wisdom method the presentation entity was named view class, however we decided to change it’s name to interaction space because it reflects more appropriately the notion of a human user interacting with

- ticular task or set of interrelated tasks. Interaction space classes are responsible for the physical interaction with the user, including a set of interaction techniques that define the image of the system (output) and the handling of events produced by the user (input). Interaction space classes isolate change in the user interface of the system and often represent abstraction of windows, forms, panes, etc.
- Task class – task classes are used to model the structure of the dialogue between the user and the system. Task classes are responsible for task level sequencing, multiple interaction space consistency and mapping back and forth between entities and interaction space classes. Task classes often encapsulate complex behavior that cannot be related to specific entity classes. Thereby, task classes isolate changes in the dialogue structure of the user interface.

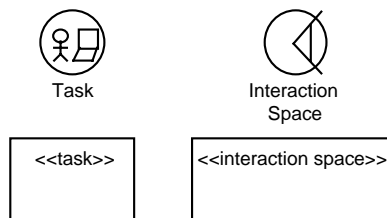


Fig. 6. Alternative representations of the interaction model class stereotypes: task and interaction space.

4.4 An application example of the Wisdom architecture

Figures 7, 8 and 9 illustrate an application example of the Wisdom architecture with different UML artifacts. The example artifacts shown throughout this section are from a simple problem definition based in similar examples worked in the literature [7] [19]. To clarify the scope of the example we cite a definition of this particular problem from [15]:

"The guest makes a reservation with the Hotel. The Hotel will take as many reservations as it has rooms available. When a guest arrives, he or she is processed by the registration clerk. The clerk will check the details provided by the guest with those that are already recorded. Sometimes guests do not make a reservation before they arrive. Some guests want to stay in non-smoking rooms. When a guest leaves the Hotel, he or she is again processed by the registration clerk. The clerk checks the details of the staying and prints a bill. The guest pays the bill, leaves the Hotel and the room becomes unoccupied."

To the left hand side of Fig. 7 we show a possible use-case structure for the above problem statement. The use-case model illustrates how different interaction and analysis classes collaborate in their realization. For example, the *customer browser* class, the *identify customer* task and the *customer* entity all collaborate in the realization of the three use-cases depicted in Fig. 7. In the middle of the illustration are interaction classes (interaction space and task classes). The task classes accommodate the dialogue of the interactive system, hence, for example, *check avail-*

ability and *identify customer* are high-level tasks belonging to the dialogue structure of this particular architecture. Task objects also guarantee consistency between interaction spaces, one example of such responsibility is visible between the *confirm reservation* task class and the *customer browser* and *customer reservations* interaction spaces. To the right hand side of Fig. 7 is the analysis model. In this example there are no boundary objects to interface external systems.

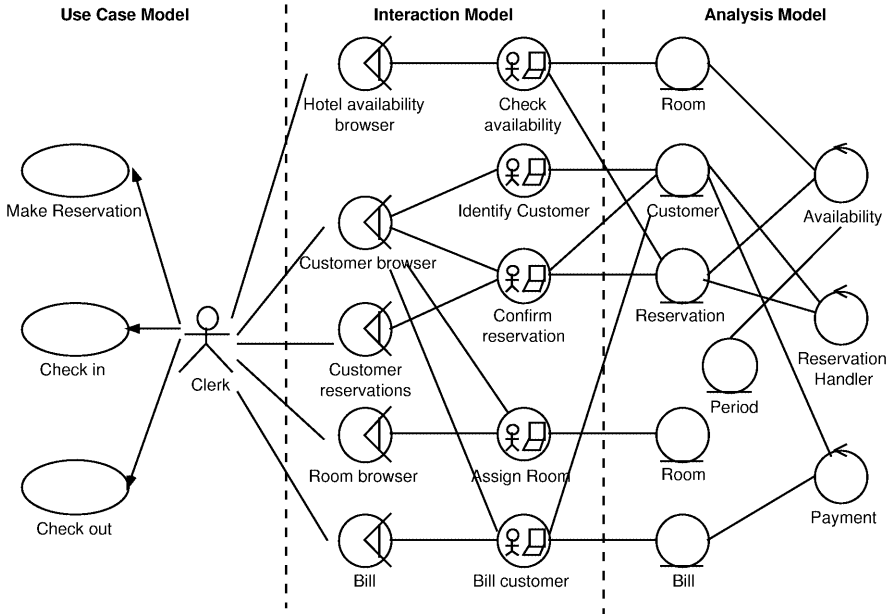


Fig. 7. Example of a use-case model, an user interface architecture and an internal architecture for a simple Hotel Reservation System.

Fig. 8 presents a hypothetical “traditional” solution (i.e., following the existing UML analysis framework). This solution was adapted from the practice illustrated in other examples in the literature (e.g., [4] [20]). Comparing both architectures we can draw some conclusions:

- The usage of boundary classes in traditional approaches, tends to follow the same functional decomposition of use-cases, i.e., there is usually a 1-to-1 mapping between use-cases and boundary classes. Such decomposition is less adaptable to change, compromises the reuse potential of the UI compromising the consistency and coherency of the UI (a major usability factor). The practical consequences of this problem are evident in the example provided. For instance, the *Customer browser* interaction space in the new approach realizes all three use-cases, such responsibility is spread between the three boundary classes in the “traditional” approach. Therefore, there is a greater probability of increased inconsistency in the traditional approach. One can always argue that such considerations should only take place at lower levels of abstraction (e.g. design); but preventing such decisions at design time is the whole purpose of devising the architecture in the first place;

- In “traditional” approaches boundary classes usually presuppose a user-interface technology or style (window, screen, etc.). Although one can argue that this is not explicitly a problem with the architectural framework, we believe it’s closely related to the absence of user interface specific architectural elements (either presentational or dialogue). For instance, the lack of lower level notational extensions supporting the reification of the presentation aspects of the UI is clearly one factor preventing technology free architectural representations;
- Another important problem, visible in both examples, is related to the placement of user interface specific logic. In the “traditional” approach, such logic can only be attributed to control or boundary classes, therefore, UI specific functionality either is bounded to presentational aspects (boundary) or to internal functionality (control). This limitation violates the principle of the separation of concerns. The result is a structure less adaptable to changes and reuse. In addition, there are clearly problems deploying such architectural models into implementation framework, for instance, a 3-tier physical implementation model very common in mainstream client-server applications. Contrasting the two examples provided, there is a clear mapping in the new framework between tasks and client side components and controls and server-side components.

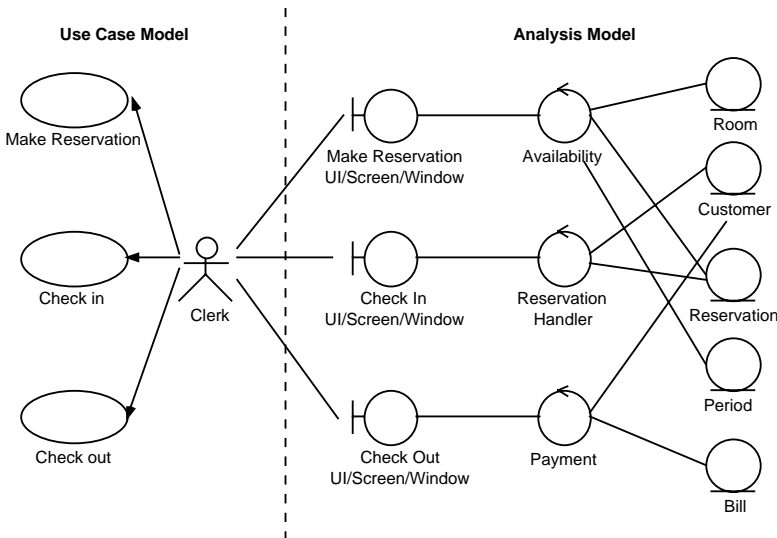


Fig. 8. Example of an hypothetical analysis architecture for the hotel reservation system problem

The following two artifacts (Fig. 9 and 10) illustrate the reification of parts of the architecture in Fig. 7. The artifacts, shown below, typically belong to design level models, in this particular case part of the dialogue model (Fig. 10) and part of the presentation model (Fig. 7). It is not our remit here to present and discuss the notation of dialogue and presentation models, refer to [21]. Alternative notations could be used to realize the analysis model, for example, the task hierarchy in Fig. 9 could be ex-

pressed through UML activity diagrams. In Fig. 9 we use an UML based notation similar to the one used by ConcurTaskTrees [2]. This notation expresses temporal dependencies between tasks through UML constraints, task decomposition is accomplished through aggregation relationships. Analogously, Fig. 10 illustrates a possible notation for the presentation model. Containment relationships between interaction spaces are shown as aggregations and navigation between interaction spaces as <<navigate>> relationships (stereotypes of UML associations).

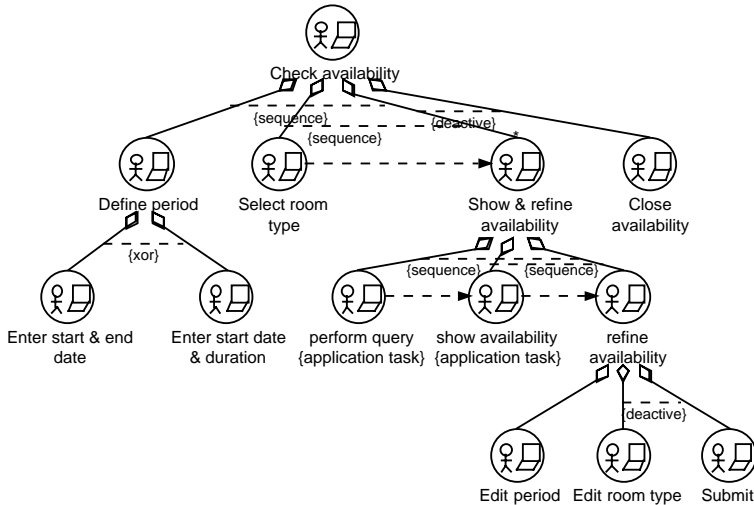


Fig. 9. Example of a Wisdom task model for the check availability top level task.

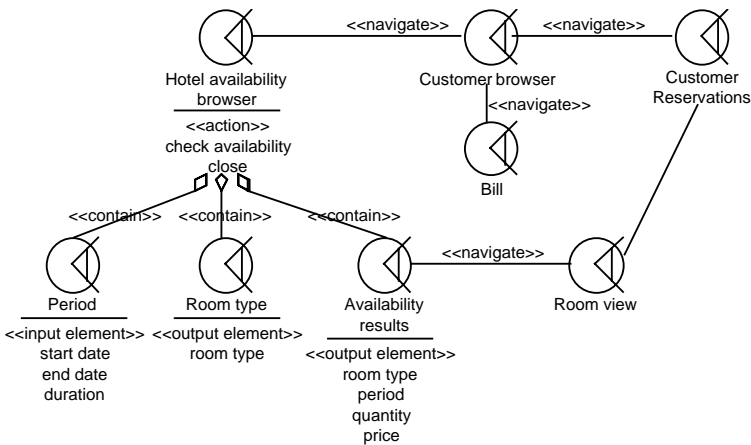


Fig. 10. Example of a Wisdom presentation model for the hotel availability browser

5 Conclusions

In this paper we discussed how the UML could be used to successfully represent conceptual architectures for interactive systems. We presented different successful approaches for conceptual and implementation architectures for interactive systems and discussed the problems with the system-centric nature of the analysis model in the Unified Process approach, also suggested as the UML standard for software development processes.

The Wisdom architecture presented here is actually an evolution of the Unified Process information space to include the dimensions of presentation and dialogue, well known to the interactive system architecture field. Our proposal accomplishes two dimension spaces that leverage both on object-oriented software engineering and user interface design knowledge. That way, the Wisdom architecture is a model that supports co-evolutionary development of models and artifacts from both worlds, leveraging collaboration, artifact change and tool support.

References

1. Artim, J. Integrating User Interface Design and Object-Oriented Development Through Task analysis and Use-cases. in CHI'97 Workshop on Tasks and Objects, 1997.
2. Paternò, F., Model Based Design and Evaluation of Interactive Applications, Applied Computing, London: Springer-Verlag, 1999.
3. Kovacevic, S. UML and User Interface Design. in UML'98, Mulhouse - France, 1998.
4. Jacobson, I., G. Booch, and J. Rumbaugh, The unified software development process, The Addison-Wesley object technology series, Reading, Mass: Addison-Wesley, 1999.
5. M. Goldberg, D.R., Smalltalk-80: The language and its implementation: Addison-Wesley, 1983.
6. Coutaz, J. PAC: An object-oriented model for dialogue design, in INTERACT'87, Elsevier Science Publisher, 1987.
7. D. Duce, D.H., M. Gomes, ed. User Interface Management and Design, Springer Verlag, 1991.
8. Coutaz, J., Software Architecture Modeling for User Interfaces, in Encyclopedia of Software Engineering, Wiley, 1993.
9. Pfaff, G. and P.J.W.t. Haguen, eds. User Interface Management Systems., Springer-Verlag: Berlin, 1985.
10. Bass, L., A metamodel for the runtime architecture of an interactive system: The UIMS developers workshop, SIGCHI Bulletin, 24(1): p. 32-37, 1992.
11. Jacobson, I., Object-oriented software engineering: a use case driven approach, New York: ACM Press - Addison-Wesley Pub, 1992.
12. Kruchten, P., The Rational Unified Process: an Introduction, Object Technology Series: Addison-Wesley, 1998.
13. OMG, Unified Modeling Language 1.3, Object Management Group, 1999.
14. Nunes, N.J. and J.F.e. Cunha, Whitewater Interactive System Development with Object Models, in Object Modeling and User Interface Design, M.v. Harmelen, Editor, Addison-Wesley, to appear.

15. Nunes, N.J., et al., Interactive System Design with Object Models (WISDOM'99), in ECOOP'99 Workshop Reader, A. Moreira, S. Demeyer, Editors, Springer-Verlag, 1999.
16. Constantine, L.L. and L.A.D. Lockwood, Software for use : a practical guide to the models and methods of usage-centered design, Reading, Mass.: Addison Wesley, 1999.
17. Mark van Harmelen, et al., Object Models in User Interface Design. SIGCHI Bulletin, 29(4), 1998.
18. Artim, J., et al., Incorporating work, process and task analysis into industrial object-oriented systems development. SIGCHI Bulletin, 30(4), 1998.
19. Dayton, T., A. McFarland, and J. Kramer, Bridging User Needs to Object Oriented GUI Prototype via Task Object Design, in User Interface Design, L.E. Wood, Editor, CRC Press: Boca Raton - Florida - EUA, 1998.
20. Conallen, J., Building Web Applications with UML. Object Technology Series: Addison-Wesley, 1999.
21. Nuno Jardim Nunes, J.F.e.C. Towards a UML profile for interaction design: the Wisdom approach. in Proc. UML'2000, York - UK: Springer-Verlag LNCS, 2000.