FROM CONCEPTUAL MODELLING TO ARCHITECTURAL MODELLING — A UCD METHOD FOR INTERACTIVE SYSTEMS

QINGYI HUA , HUI WANG , CLAUDIO MUSCOGIURI , CLAUDIA NIEDERÉE , AND MATTHIAS HEMMJE *

Abstract. Software design is a process of transformation from problem domain to implementation domain based on two crucial models: conceptual and system models. User-Centred Design (UCD) differs from traditional software design in the perspective providing to conceptual modelling. It concentrates on knowledge about the context of use rather than the accidental features of problem domain. UCD is also concerned with the integration of that knowledge into the system model by means of contextualization that allows combining the descriptions of usage with the functional specifications during the process in order to accomplish a valid design solution. In this paper, we present the ADOI (Another Dimension of Information) approach that aims at providing support for contextual development. Due to its declarative specifications ADOI allows explicit conceptualization of usage, as well as of contextual linkage required for the transformation. A conceptualization- driven architecture is in ADOI open with respect to different perspectives for the user interface and the system. As a result, ADOI realizes the role of a complement of existing methods by providing a development support that can be integrated into different design models.

Key words. User-centered design, conceptual models, Architectural models, Contextual development, Software design processes

1. Introduction.

A usable interactive system provides its users with useful concepts for solving their problems at hand without becoming bogged down in accidental features of problem domain [1]. Such usability is difficult to be achieved by a traditional software process since (users') *goals and system state differ significantly in form and content, creating the gulfs that need to be bridged if the system can be used* [2]. To span the gulfs of usage, an increasing number of people from Human-Computer Interaction (HCI) and Requirements Engineering (RE) communities are getting a consensus to applying a user-centred perspective for the software design, or user-centred design (UCD) throughout the software process (e.g. [3], [4], [5]).

It should be aware that UCD is a complement to existing design methods rather than a replacement for them according to ISO 13407 standard [6]. In other words, UCD is to model knowledge about the context of use and to integrate that knowledge into existing design models rather than to develop a particular set of models. Contextual development has been recognized to be crucial for meeting the demands of user-centred systems design [5], in which the gulfs are spanned with a process of transformation from usage domain to implementation domain. However, it is still lacking an effective way of contextualization allowing partitioning and packing different perspectives within current UCD approaches, resulting in only less than 30% of them were used in actual projects in accordance with a recent study [3].

Blum [7] identifies that two models — conceptual and system models — are commonly used in the traditional software processes for the transformation. The conceptual model is representation of the problem raised by the users and is carried out for the purpose of understanding, while the system model (more recently software architecture [8]) represent a starting point of the software solution for the problem.

Conceptual modelling plays a central role to capture system's requirements. Rolland [4] identifies two sub-types of requirements that have to be modelled. User-defined requirements arise from people in the organization and reflect their goals,

*Fraunhofer IPSI, Dolivostr. 15, D-64293 Darmstadt, Germany (hua, hwang, muscogiuri, niederee, hemmje@ipsi.fhg.de).

intentions and wishes. Domain-imposed requirements are facts of nature and reflect domain laws. This means that the universe of discourse has to be partitioned into two parts: *the usage world* and *the subject world* [9]. The usage world describes the context in which the system is to be used, or the context of use and consists of the characteristics of the intended users, their tasks, and the environment. The subject world describes the context in which the system is to be set up, and consists of real world objects and behaviours. We argue that there is a third sub-type of requirements, that is, the contextual linkage that represents the semantic relationships between the two worlds, which begins to bridge the gulfs between usage and subject features.

There is a third world, that is, *the system world* [9] that is the world of architectural models in which the requirements mentioned above must be addressed. Traditionally, the architecture of an interactive system has been conceived as its organization of a presentation and functional layer as a collection of interacting components. The components of the functional layer, also called function core (FC), realize knowledge related to the subject domain that the system is intended to provide. The components of the presentation layer, also called user interface (UI), are responsible for users viewing and controlling concepts and relationships related to their tasks. Coutaz [12] identifies a third layer, a mediator that specifies a protocol for control strategy and data exchange between the UI and the FC. The role of the mediator is to act as a contextual mechanism that represents the state of the FC in terms relevant to the user tasks but the representation is still presentation-independent, and transforms the user tasks into the actions of the FC. Modelling the mediator explicitly bridges the gulfs further towards a valid software solution, which requires the knowledge about the contextual linkage.

Obviously, UCD requires both of the contextual linkage between the two worlds and the mediator that deliver the linkage to design models. In this paper, we present the ADOI (Another Dimension of Information) approach that aims at providing support for contextual development. Due to its declarative specifications ADOI allows explicit conceptualization of usage, as well as of contextual linkage required for the transformation. ADOI contains conceptualization-driven architecture to cope with knowledge allocation with respect to functionality of the UI and the FC, as well as the contextual linkage between them. As a result, ADOI realizes the role of complement by providing a development support with the integration of different design approaches.

This paper is structured as follows. We identify first which concepts in the two worlds are required for specifying relationships between the users' goals and the system functionality in section 2. We introduce our approach and demonstrate an example in section 3. Finally the conclusions are presented in section 4.

2. Conceptual models for contextual development. A conceptual model is a collection of concepts and their relationships, which embodies people's shared understanding for some perspective with respect to some domain of interest. The traditional way of conceptual modelling is to understand and represent features (i.e. entities and behaviours) of problem domain from a system's perspective. It concentrates on what the envisioned system should do, or on its functionality by means of domain and functional models (e.g. object and use-case models in UML [13]). These models are unable to model users' knowledge about problem-solving goals, although they are necessary for modelling the information maintained and the functionality provided by the system.

Instead of modelling the concepts of entities and behaviours in the problem domain, conceptual modelling in UCD explores the users' goals and their activities to meet these goals in order to achieve a usable system. To provide a user-centred perspective, various task-based models have been proposed for the representation of these activities. For example, some approaches have proposed goal-based use-case models, e.g. [10], [14]. Unlike use-case model in UML, goal-based models can be qualified as user-centred since goal cases are described around goals with respect to users' concepts. When considering the process of transformation, however, it turns out that *a use case is delivered for a given set of features* and *the features cross multiple use cases, making the tracking more complicated* [10]. To span the semantic distance between the goals and features, a goal case must be refined as a set of traditional use cases before the architectural specification can be specified. This is typically not a user-centred way. On the other hand, task models in HCI (e.g. [11]) demonstrate a similar problem. They attempt to decompose a task down to the primary forms of the task in order to provide a connection between the task and the actions that realize the task in the problem domain.

In fact, the rationale of contextual development requires a combination of both perspectives. On the one hand, it requires explicating the meaning of a user task with the users' concepts. On the other hand, it also requires connecting the meaning with the system's concepts. It implies that objects in the information space with respect to the universe of discourse have to be partitioned into two sub-types: conceptual objects in the usage world, and domain objects in traditional sense in the subject world. Conceptual objects in this sense represent the state of the domain objects in concepts meaningful to the users.

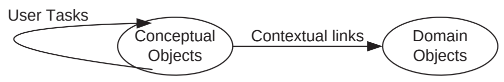

Fig. 2.1. *The relationship between the two perspectives*

As shown in Fig. 2.1, in this paper we define a use task is a mapping relation over conceptual objects rather than over domain objects, whereas a contextual link specifies a collection of mapping relations, or semantic operations, from conceptual objects to domain objects. These semantic operations interpret the user task over the domain objects and thus, the semantics of user tasks are declarative both for the users and developers. In this way, we can describe a goal as a state of conceptual objects that a user wishes to achieve in the future. Hence, the meaning of the corresponding task can be mapped to the state. We can also discuss tasks more abstractly without becoming bogged down to implementation details by dealing with state transitions.

In the remainder of this paper, we use goal cases to specify tasks and goals in terms of the definition, in which each of user inputs and system outputs is specified over the conceptual objects. We also use semantic mappings from the conceptual objects to the domain objects, which links the two sets of objects contextually. As a result, the task performance could be conceived of as a transition from the current state of conceptual objects to the goal state. Actually it is the application of the semantic mappings over domain objects, which makes the transition.

3. The ADOI approach. The ADOI approach aims at supporting the process of contextual development from conceptual models to architectural model to realize the users' goals in a declarative way.

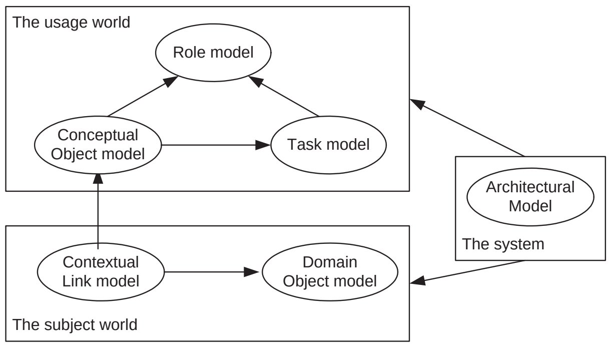

Fig. 3.1. *The ADOI framework of models*

As shown in Fig. 3.1, the ADOI approach provides a framework comprising the models for the understanding and representation of the corresponding worlds that the models are assumed to support. The purpose of the framework is to explicate the relationships among the models under the assumptions over each component of the models. The arrows in Fig. 3.1 show the dependency relationships between the models, which means that a model has to be changed if a model it depends on is changed.

In the following we discuss the concepts and their relationships modelled by these models. We demonstrate the usage of these models with a simple hotel reservation example in [15].

3.1. Role model. Conceptual modelling for the usage world means understanding and modelling the people who are involved in affairs. For understanding the usage, the roles that users play can be more important than the users themselves [14]. A role represents a collection of common features abstracted from actual users who might interact with the system with similar goals. Unlike an active object, the characteristics of a role rely on its psychological aspects. A role can have attributes that specify knowledge, skill, experience, education, training, responsibility, etc. A role can also have behaviour when interacting with the system. As a result, a role is not only *a meaningful collection of tasks performed by one or more agent* [11]. It is also related to a set of conceptual objects that reflect the role's understanding of the usage.

A role model in ADOI is composed of a collection of roles and the relationships between them. A role is a collection of goals that represent *the state the person wishes to achieve* [2]. Roles can be involved in a type hierarchy that specifies the generalization of goals.

Fig. 3.2 illustrates the main artefact of the role model for the hotel reservation example. It shows the roles involved in the affair of reservation. The association between the customer and the clerk is many to one, implying the clerk can serve multiple customers, but a customer is assigned to one clerk.

3.2. Task model. A significant activity in conceptual modelling is to analyse, identify and specify tasks to be performed by roles. Task modelling for contextual development requires this activity to be centred on the users' goals, and later the meaning of the tasks is interpreted from a system's perspective, both in a declarative way.

In ADOI, a distinguishable characteristic is to separate the description of tasks

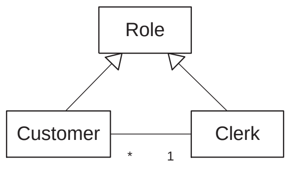

Fig. 3.2. *A role model for the example application*

from their references to domain features by introducing a task model and a contextual link model. Achieving the separation depends on our definition of goals and tasks in section 2. We assume the state of the envisioned system is determined in terms of conceptual objects rather than of domain objects. The role of contextual links is to map the state of conceptual objects to domain objects. As a consequence, tasks are performed over conceptual objects rather than domain objects.

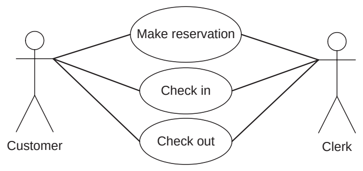

Fig. 3.3. *A use-case model for the representation of user tasks*

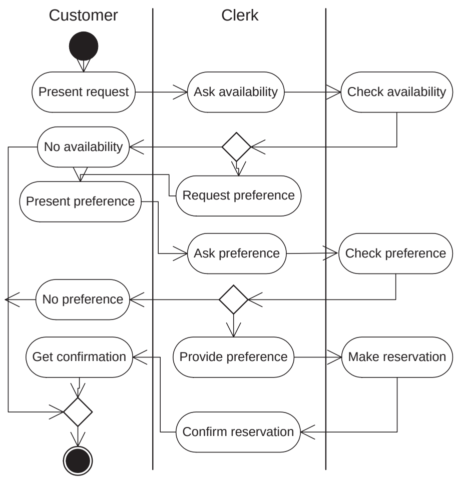

Fig. 3.4. *A workflow for the 'make reservation' task*

ADOI uses a goal-case model to describe tasks and their relationships. However, our goal cases refer directly to the state of relevant conceptual objects rather than to the one of domain objects that suffers from the problem with semantic distance as mentioned earlier. Such problem will not appear in our case according to the task definition. In this way, we can reuse the original form of use-case model in UML without any extensions and/ or constraints, but on a consistent level towards the users' goals with respect to the users' concepts.

A use case in ADOI includes a name specifying a task, and a collection of attributes, such as goal, pre-/post-condition, significance, frequency, etc. A use case can be in a generalization hierarchy. Use cases can have ‘ include’ and ‘extend’ relationships [13]. Fig. 3.3 and Fig. 3.4 illustrate the main artefacts of the task model for the hotel reservation example. Fig. 3.3 shows the use-case diagram that specifies the tasks and their relationships with the roles. Fig. 3.4 shows an activity diagram detailing the ‘make reservation’ use case. Verbs in Fig. 3.4 are used for specifying the customer and the clerk tasks, and the system responses. Nouns in Fig. 3.4 are used for specifying conceptual objects. It is easy to see that the use case is described from a user’s perspective, and maintain a linear relationship over the conceptual objects. The system responses in Fig. 3.4 provide then meanings for the clerk’s tasks from a user’s perspective and, hence, have to map to domain features, which we will discuss in section 3.5.

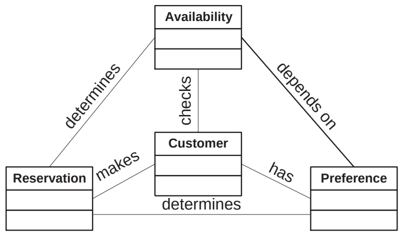

FIG. 3.5. *A conceptual object model for the ‘make reservation’ task*

3.3. Conceptual object model. The conceptual-object model in ADOI contains objects and relationships among the objects. A conceptual object is an intangible thing in a user’s mind and, hence, belongs to the usage world. Conceptual objects are often closely related to user tasks. As a result, modelling tasks and discovering conceptual objects is an iterative process.

Fig. 3.5 shows a class diagram for the ‘make reservation’ use case. The conceptual-object model contains classes that are significant from a user’s perspective. Some of these classes may not be realized in the system design. For example, the customer’ class is important from a clerk’s perspective, but it may be an attribute of the class ‘reservation’ in the system design. This is similar to the relationship between a domain model and the system design.

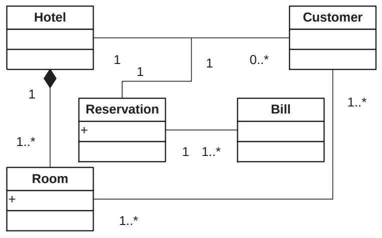

FIG. 3.6. *A domain object model for the example application (adopted from [15])*

In general, conceptual objects represent a user’s understanding of the usage. Provision of task specification with conceptual objects is important for the users to understand and agree on their tasks. Conceptual objects are often aggregations from real

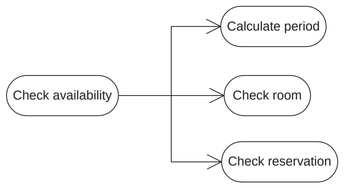

FIG. 3.7. *A contextual link model for the 'Check availability'*

things in the domain by reorganizing information provided by these things.However, it is difficult to discover the information for the restructuring and reorganization using an inside-out perspective, as proposed by [12]. Modelling conceptual objects certainly provides an effective way for this. On the other hand, conceptual objects help UI designers to concentrate on towards goal-based presentation.

3.4. Domain object model. A domain model represents the understanding of domain-imposed requirements from a system's perspective. In ADOI, A domain model does not contain objects explicitly related to user tasks. In other words, the domain model comprises pure objects in the subject world. Fig. 3.6 shows the domain model in the example application.

The benefit of using a pure domain model is to reuse and share it among different applications in the same domain since these applications can be envisioned for different requirements. In other words, it facilitates the development of *domain ontology* [4] for the purpose of interoperability between the applications.

3.5. Contextual link model. It is important to understand and identify existing relationships from the usage world to the subject world for tracing domain features. This means that the specified tasks for the envisioned system in the use cases have to be mapped to corresponding domain features.

ADOI attempts to specify the relationships in a declarative way by a contextual-link model. A contextual link maps a user task defined on the conceptual objects to a collection of semantic operations defined on the domain objects. The role of contextual links can be two-fold: tracing domain objects that are relevant to tasks; and identifying operations over domain objects that may be performed by the tasks.

In this moment, we do not consider how a task invokes corresponding operations (synchronous, or asynchronous) and how the operations will be performed (sequential, or concurrent). These how questions are delayed until the design time.

For example, Fig. 3.7 specifies a contextual link, which map the ' check availability' task to three operations. The contextual link identifies not only the operations, but also a 'period' object that does not exist in the domain model. In fact, the contextual link model can be used as a means for reasoning about why a domain object is required, and what are its attributes.

3.6. Architectural model. The role of an architectural model for contextual development is to act as a means to deliver knowledge about user tasks to the system design, as we mentioned earlier. Fig. 3.8 presents the assumed knowledge space realized by the ADOI architectural model. The space is composed of two subspaces:

1. *Functional space.* Functional space represents knowledge about the control and process of domain-specific information, and about the other non-human active agents.

2. *Interaction space.* Interaction space covers knowledge about the presentation and control of human-specific concepts.

The dimensions in each subspace are independent of each other. This is a necessary condition for a declarative specification. For example, if a shared information space is used both for the functional knowledge and the interaction knowledge, as proposed by [15], the dialogue cannot be independent of the behaviour and, thus, the behaviour dimension has to be reified in terms of the particular characteristics of the dialogue.

The shared dimension between the two subspaces is the concept dimension of the interaction space and the behaviour dimension of the functional space. The dimension connects the two subspaces together by means of the process of human-specific concepts through domain-specific information.

The knowledge space is a contextual extension to UML architectural framework specified by the analysis profile [13]. In fact, the UML profile only contains the functional space in the Fig. 3.8, reflecting a system-centred perspective.

ADOI architectural model defines six stereotypical classes:

1. Task-boundary classes. A task-boundary class is assigned to knowledge about a user as a role with respect to both functional and non-functional aspects. Therefore, it must facilitate a user to create user conceptual model [2] that represents the user's understanding of a system. Task-boundary classes model interaction between the system and its human actors. They represent interaction contents that reflect the presentation of the user's concepts, and that request information from the user, rather than abstractions of physical user-interface components [15].

2. Task-control classes. A task-control class is assigned to knowledge about user tasks as multiple mapping relations with respect to functional respects. Task-control classes are responsible for sequencing the interaction between the user and the system on the task level, and for transferring information between task-boundary and system-entities via task- entities. They often encapsulate special behaviour related to user task, so that it isolates change of the structure of dialogue.

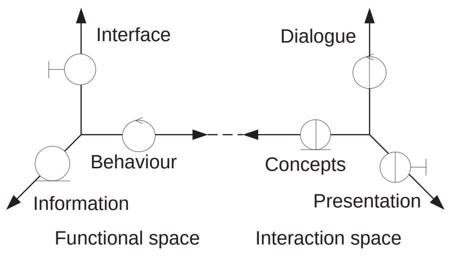

Fig. 3.8. *The ADOI knowledge space*

1. Task-entity classes. A task-entity class is assigned to knowledge about a conceptual object particular for the context of a task with respect to non-functional aspects. Task-entity classes are responsible for the interoperability between task-control and the system-control classes, such as temporal coordination and data exchange. They represent short-term information, and work as a working memory that specifies the state of the system in terms meaningful of the users and independent of the presentation of the system.

2. System-control classes. A system-control class is assigned to knowledge about

the contextual relationships between user tasks and system behaviours. System-control classes often cope with complex control logic related to domain data. They often encapsulate special behaviour related to system tasks, so that they isolate change in the structure of data.

3. System-entity classes. A system-entity class is assigned to knowledge about a domain model as a collection of entities with respect to the functional aspects. System-entity classes represent information that is long-lived. Their role is much similar to the 'entity class' in the UML analysis profile [13].

4. System-interface classes. A system-interface class is assigned to knowledge about the relationships between the system and its environment. Their role is much similar to the 'interface-class' in the UML analysis profile [13].

Fig. 3.9 demonstrates an artefact of architectural modelling for the 'make reservation' use case in terms of the conceptual models built for the example application. Objects in Fig. 3.9 are specialized from the stereotypical classes, e.g. the 'Availability' object is an instance of the task-entity class that realizes the corresponding user's concept, whereas the 'Availability' object is assigned to the functionality to present the 'Availability'.

Task-entity objects in Fig. 3.9, in fact, act as the mediator between the UI and the FC. They realize the contextual links between task- control objects and system-entity objects. The mediator works also as a short-term memory for domain-knowledge delegation. For example, the 'Availability' object in Fig. 3.9 stores information related to the possibilities of the preference. As a result, the 'Availability' object is first checked for the preparation of the preference, avoiding the repetition of work that might be time- consuming on the domain level.

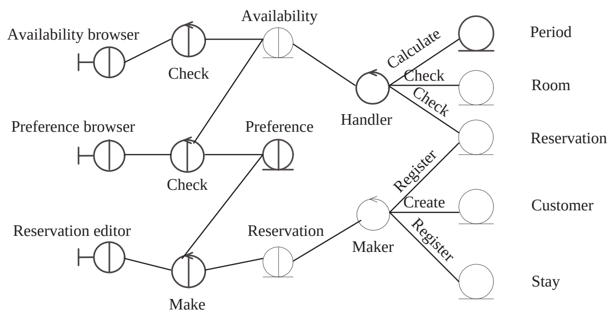

Fig. 3.9. *An analysis model for the 'make reservation' use case*

4. Conclusions. In UCD conceptualization and contextualization should be considered equally important for creating a usable and valid design solution. Models in ADOI have been built in this way. Static and explicit concepts defined in those models allow a declarative specification for the usage, and the integration of different perspectives through the contextual linkage. A novel conceptualization-driven architecture is provided with knowledge allocation with respect to functionality and contextual linkage. Consequently, a seamless development with the integration of different design approaches is systematically supported.

REFERENCES

[1] Hua, Q., Wang H., Hemmje, M., *Conceptual modelling for interaction design*, Will appear in the Proceedings of the 10th International HCI Conference. Crete, Greece(2003)

[2] Norman, D., *Cognitive engineering, In: Norman, D. and Draper, S (eds.): User centered system design*, Lawrence Erlbaum Associates, Hillsdale, NJ. (1986) 31-61.

[3] Hudson, W, *Towards unified models in user-centred and object- oriented design*, In: M. van Harmelen (eds.): Object Modeling and User Interface Design, Addison-Wesley (2000).

[4] Rolland, C. and Prakash, N., *From conceptual modelling to requirements engineering*, Annals of Software Engineering, 10(2000) 151-176.

[5] Stary, C., *Contextual prototyping of user interfaces*, proceedings of ACM Dis'00, (2000) 388-395.

[6] ISO/TC159, *Human-centred design process for interactive systems*, Report ISO 13407: 1999. Geneva, Switzerland (1999).

[7] Blum, B.I., *Beyond programming*, Oxford University, N.Y. 1996.

[8] Shaw, M, Garlan, D., *Software architectures: perspectives on an emerging discipline*, Pretice-Hall, Englewood Cliffs, NJ, 1996.

[9] Jarke, M. Pohl, K. and Rolland, C., *Establishing visions in context: towards a model of requirements processes*, Proc. 12th Intl. Conf. Information Systems, Orlando, Fl, (1993).

[10] A. Cockburn, *Structuring Use Cases with Goals*, JOOP/ROAD 10(5) Sep'97 and 10(7) Nov'97.

[11] Puerta, A. and Einsenstein, J., *Towards a general computational framework for model-based interface development systems*, ACM CHI'97, (1997).

[12] Coutaz, J. and Balbo, S., *Applications: A dimension space for user interface management systems*, The proceedings of ACM CHI'91, (1991) 27-32.

[13] G. Booch, et al., *The Unified Modelling Language User Guide, Reading*, MA: Addison-Wesley, 1999.

[14] Constantine, L., *Use cases for essential modeling user interfaces*, ACM interaction, 4 (1995) 34-46.

[15] Nunes, N. and Cunha, J. F.: Wisdom, *A software engineering method for small software development companies*, IEEE Software, September/October, (2000) 113-119.