A UCD Method for Modeling Software Architecture

Qingyi Hua, Hui Wang, Claudio Muscogiuri, Claudia Niederée, and Matthias Hemmje

Fraunhofer – IPSI
Dolivostr. 15, D-64293 Darmstadt, Germany

{hua, hwang, muscogiuri, niederee, hemmje}@ipsi.fhg.de

ABSTRACT

User-centred design (UCD) is a complement to software design approaches rather than a replacement for them. It implies that UCD concentrates on the process of modelling knowledge about usage of a system rather than on representation of technical features and constraints particular for software design. However, there still lacks a seamless support for the transformation process from usage to design because of the non-linear property of the process. 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 perspective for the user interface and the system. As a result, ADOI realizes the role of a complement by providing a development support that can be integrated with different design approaches.

1. INTRODUCTION

To achieve usable systems, an increasing number of people from the Human-Computer Interaction (HCI) and Requirements Engineering (RE) communities are getting a consensus to user-orientation [1, 2, 3]. Knowledge about the context in which the systems will be used (e.g. users, and their tasks) is considered equally important for guiding the process of software development, as the knowledge about the features of the problem domain. However, a recent study [3] reveals that only less than 30% User-Centred Design (UCD) approaches were used in actual projects. Furthermore, many people involved in the projects acknowledged that they had integrated different methods and models in their processes, because of the lack of mature development support in these UCD approaches.

According to ISO 13407 standard [4], in fact, UCD is *complementary to existing design methods and provides a human-centred perspective that can be integrated into different forms of design process in a way that is appropriate to the particular context*. It implies that UCD concentrates on modelling knowledge required for achieving problem-solving goals instead of technical features and constraints that are particular for a class of design models. However, the latter comes into play when transforming knowledge content into the appropriate forms for the existing design methods. To provide an effective support for the transformation, we argue that *software architecture typically plays a key role as a bridge between requirements and implementation* [5] since it is required for delivering contextual knowledge to design models but still in technology-independent way.

The architecture of an interactive system can be grossly 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 (entities, actions) related to the problem 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 rather than directly to domain features. Coutaz [6] has identified 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 working memory 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.

It is difficult to ensure the architectural specification for the mediator in technology-independent way without contextual knowledge, because of the significant difference between the functional assumptions made for the UI and those made for the FC. The contextual knowledge is to be found outside the system itself since it is a kind of non-functional requirements derived from usability. This suggests establishing a contextual linkage between usage and domain.

Rolland [7] identifies two sub-types of requirements: user-defined requirements that arise from people in the organization and reflect their goals, intentions and wishes; domain-imposed requirements that are facts of nature and reflect domain laws. This means that the universe of discourse has to be partitioned into two parts, that is, *the usage world* and *the subject world* [8]. The usage world describes the context in which the system is to be used, or *the context of use* [4] and consists of the characteristics of the intended users, the tasks the users are to perform, and the environment in which the users are to use the system. The subject world describes the context in which the system is to be set up, and consists of real world objects that are to be represented in the FC. We argue that there is a third sub-type of requirements, that is, the contextual linkage between the two worlds.

The traditional way of engineering systems is through conceptual modelling that generates a specification for the design of the envisioned system. Conceptual modelling [11] traditionally is used to understand and represent features related to the subject world. In order to model the usage world, a number of goal- and task-based approaches have been proposed. However, the contextual linkage is still lacking.

For instance, when considering the goal-based use-case model for architectural modelling, it turns out that *a use case is delivered for a given set of features* (in the subject world) [9]. As a result, the use 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. Task-based approaches demonstrate a similar problem e.g. [10]. Instead of modelling the contextual linkage, 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 implements the task in the subject world.

UCD requires both of the contextual linkage between the two worlds and the mediator that deliver the linkage to design models because of the non-linear transformation process from usage to design. 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.

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 in section 4 we conclude our work by indicating the work in the future.

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 aspect with respect to some domain of interest. Traditionally, the purpose of conceptual modelling is to generate a specification of the envisioned system. Recently the notion of conceptual modelling has been augmented for specifying the relationships between the system and the environment in which the system is to be used. Although there may be alternatives, the basic types of conceptual models in use for contextual development can be classified into the following categories:

- Task model. A task model is a specification of tasks to be performed with the system and the relationships among these tasks. Task models can, for example, be hierarchical task models, traditional use cases, or goal-based use cases.
- Domain model. A domain model defines a collection of objects in the environment that the system will be used in and their relationships. A domain model specifies information to be maintained by the system. It usually contains real things in the environment. Conceptual objects in the users' minds may be included into the domain model, depending on which perspective is in use.
- Role model. A role specifies a typical user who performs tasks with the system and, hence, can be characterized as a set of tasks. A role model represents the different types of roles and their classification.

In this section, we discuss the characteristics of task models because of their crucial role for contextual development and argue why the contextual linkage between the users' concepts and domain concepts is required for contextual development.

2.1. Declarative models

A task can be conceived as an input-output relation over objects; task applications specify state transitions. There are two different approaches to modelling the semantics of tasks:

- Declarative semantics. With declarative semantics, a task is given meaning by mapping it to well-known concepts. The task can be well understood without reference to any specific computational procedure for its realization.
- Procedural semantics. In procedural way, a task is given meaning by referencing a real or virtual

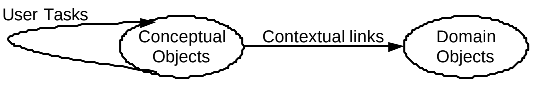

Figure 1. The relationship between the two perspectives

procedure that specifies the state transitions. To obtain the meaning is to simply execute the procedure and observe the outcome.

There is another dimension of perspective relevant to a declarative task model with respect to contextual development, that is, to whom a task can be well understood:

- User-centred. A task is defined with the users' concepts. However, it may be not well defined for the developers because it states neither what something does in the developers' language, nor how something is done. For instance, '1 + 1' is the simplest task that has a well-understood meaning '2' for people, but it must map to the developers' concepts before it can be used for design. The mapping creates contextual relationships from the users' concepts to the developers' concepts.
- System-centred. A task is specified with the developers' concepts. For example, '+ (1, 1)' is well understood in the LISP domain, but people outside the domain must perform a learning procedure or really execute the expression with a LISP environment before they can understand the meaning of the expression.

The rationale of contextual development requires a combination of both perspectives. On the one hand, it requires a user's perspective for explicating the meaning of a user task with the users' concepts. On the other hand, it also requires a system perspective for explicating the meaning of a user task with the developers' 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. Domain objects in this sense represent the export state of the conceptual objects in concepts meaningful to the developers. However, current approaches do not make the partition. For example, domain models in HCI usually contain the both sub-types of objects, whereas object-oriented approaches do not acknowledge the existence of conceptual objects.

As shown in Fig. 1, a user task defines a mapping relation over conceptual 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. As a result, the semantics of user tasks are declarative both for the users and the developers.

In remainder of this section, we look at that what is still lacking in current task models.

2.2. Task models

The meaning of *task* has been specified in somewhat different senses in HCI and RE, resulting in some confusion. For example, approaches in HCI, e.g. [1], [13], define a task as an activity performed to achieve a goal. It implies that a task is a fixed method or a procedure to be undertaken in order to obtain the goal or the meaning of the task.

In goal-based RE, a goal is sometimes defined as *something that some stakeholder hopes to achieve in the future* [18]. However, it is still not articulate who is the stakeholder and what is something.

In this paper, we 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. In this way, we can discuss tasks more abstractly without becoming bogged down to implementation details by dealing with state transitions.

How to describe a task specification relies on different perspectives. In HCI, a task is usually modelled as a hierarchical structure that reifies task/subtask decomposition down to specific actions that a user may perform over conceptual and domain objects through the UI of the system. *Therefore, user-task models provide two types of information about tasks: structural and procedural* [14]. That is, a contextual link in Fig. 1 is given meaning by reference to a context-sensitive structure rather than a declarative specification for the structure. Furthermore, it is still difficult to discover the goal of the task unless the goal is already known. Hierarchical task models may be appropriate for the design of components of the UI of the system, which should come later when the architecture of the system has been determined.

Some approaches in HCI, e.g. [15], [16], use a hierarchical task model as an essential component of their ontology for the specification of the system. However, in our view their ontology might not be qualified as real ontology that defines *a formal, explicit specification of a shared conceptualization* [17].

Use-case models [19] have been widely used as task models in object-oriented approaches. The role of use cases can be seen as two-fold: discovering the tasks and/or goals, and determining about the relationships from tasks to domain objects. A use case in essence can be described as a process of interactions performed in turn by an actor (a human, or an active agent) and the envisioned system. Use-case models can be conceived as declarative specifications since the meanings of the actions performed by the actor are given by the corresponding system responses. It is generally agreed that the Jacobson's use-case model [20] is system-centred because the actions and their meanings are described using domain concepts. In fact, a use-case model in this situation corresponds to a state machine over domain objects [21, 22]: an action through the UI produces an event, which in turn triggers an operation performed for the purpose of state transition. As a result, the connection between the actions and corresponding system operations is linear and declarative from a developer's perspective.

To provide a user-centred perspective, some approaches have proposed goal-based use-case models, e.g. [9], [23]. Unlike Jacobson's use-case model, goal-based models can be qualified as user-centred since use cases are described around goals with respect to users' concepts. In principle, a use case in this situation specifies a task over the conceptual objects. However, the conceptual objects actually are not modelled in formal way in those approaches, implying that the nature of use-case model as a state machine over domain objects is not changed. This violates the linear property of state machine since the relation between a user task and corresponding domain objects is in general non-linear, as we have seen in the hierarchical task model. As a consequence, *the (domain) features cross multiple use cases, making the tracking more complicated* [9]. The reason is that contextual links are still lacking in the goal-based models.

In this paper, we model conceptual objects, and domain objects formally, both of which specify the information maintained by the system. On the one hand, the two kinds of objects model knowledge about usage of the system and about domain features, respectively. On the other hand, conceptual objects represent short-term information relevant to usage, whereas domain objects represent long-term information related to domain features in terms of an architectural view. We also use semantic mappings from the conceptual objects to the domain objects, which links the two sets of objects contextually.

We use goal-based use cases to discover tasks and goals of the tasks, in which each of user inputs and system outputs is specified over the conceptual objects. As a result, the task application 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 (Another Dimension of Information) approach is aimed at supporting the process of contextual development based on the underlying principle of UCD – from conceptual models to architectural model to reify the users' goals in a declarative way. The meaning of ADOI is two-fold: modelling conceptual objects and domain objects on the one hand, and making functional assumptions over the components of architecture for the two sub-types of objects on the other hand. The approach does not contain any assumptions for the implementation methods.

As shown in Fig. 2, 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, which we will discuss in the remainder of the section. The arrows in Fig. 2 show the dependency relationships between the models, e.g. both the conceptual model and the task model rely on role model. It means that a model has to be changed if a model it depends on is changed.

Contextual development for interactive systems with ADOI can be seen as a two-stage process:

- Conceptual modelling includes creating models (role model, task model and conceptual model) in the usage world, the domain model in the subject world based on the understanding of the users and of the problem domain, and the contextual link model for the connection between the two worlds.
- Architectural modelling produces an architectural model, in which concepts and their relationships in the conceptual models are assigned to or realized by the components of the architectural model.

ADOI does not prescribe any particular methods for the system design. Any design methods can be used for the design of components if they are considered to be appropriate.

This section discusses concepts and relationships

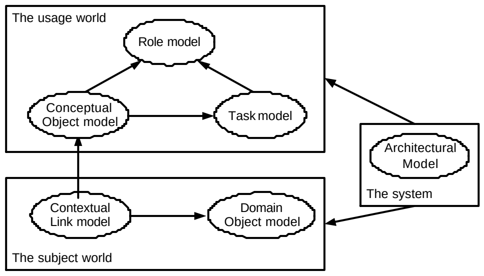

Figure 2. The ADOI framework of models

among these concepts modelled by the different models. We demonstrate the usage of these models with a simple, well-known hotel reservation example [24, 25, 26].

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* [23]. A human user plays as a role (e.g. a customer, or a clerk) as the user has a certain goal or a collection of goals to use the system. 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* [15]. It is also related to a set of conceptual objects that reflect the role's understanding of the usage. For instance, 'take photo' is a common task for any person acting as a photographer, but a beginner typically has a different set of conceptual objects from a professional when interacting with a camera. This implies that a task can be performed in different ways in terms of conceptual objects to achieve a goal.

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* [27]. Roles can be involved in a type hierarchy that specifies the generalization of goals, e.g. the beginner and the professional in the above example can be generalized as a photographer. Roles can have associations that represent semantic relationships, such as cooperation, between roles.

Fig. 3 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,

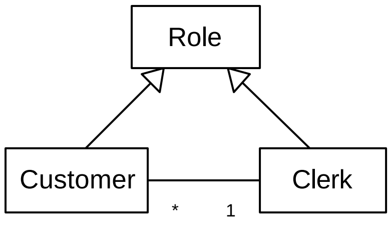

Figure 3. A role model for the example application

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 with the envisioned system. Task modelling for contextual development requires this activity to be centred on the users' goals from a user's perspective, and later the meaning of the modelled tasks is interpreted from a developer's perspective, both in a declarative way. The accomplishment of this activity relies on a shared understanding between the users and developers. In our vision such a consensus does not mean that a unified task model could be used to describe knowledge about tasks in the usage world with intertwining explicit and implicit references to the domain features in the subject world.

In ADOI, a distinguishable characteristic is to separate the description of tasks from their reference to domain features by introducing a task model and a contextual link model. Achieving the separation depends on our view about what is a task. We assume the state of the envisioned system is determined in terms of conceptual objects, or *psychological variables* [27], 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.

In ADOI, a task is defined as an input-output relation over conceptual objects. A goal is described as a state of conceptual objects that a role wishes to be achieved by the task and, thus, the result of a task can be mapped to the goal.

ADOI uses a use-case model to describe tasks and their relationships. As we mentioned earlier, use-case models suffer from a problem with the level-of-abstraction, that is, the more abstract a use case is, the further the distance is between the use case and the relevant state of the domain objects. Such problem will not appear in our case according to the task definition. Unlike many use-case models that make more extensions and/or constraints on the definition of a use case, we only add or replace several (bold) words to the original UML definition [19]:

The specification of a sequence of actions, including variants ***over conceptual objects*** *(instead of nothing here), that a system (or other entity) can perform, interacting with* ***roles*** *(instead of the original "actors") of the system.*

In this way, we can use the original use-case model without any extensions and/or constraints, but on a

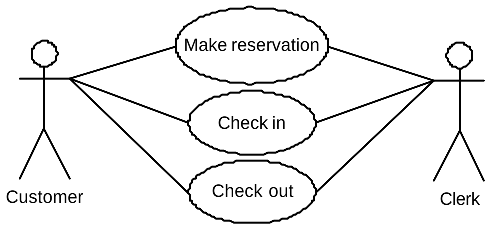

Figure 4. A use-case model for the representation of user tasks

consistent level towards the users' goals with respect to the users' concepts. It is notable that user actions are given meaning by reference to the corresponding system actions, or responses.

A use case in ADOI includes a name specifying a task, and a collection of attributers, 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 [19].

Fig. 4 and Fig. 5 illustrate the main artefacts of the task model for the hotel reservation example. Fig. 4 shows the use-case diagram that specifies the tasks and their relationships with the roles. Fig. 5 shows an activity diagram detailing the 'make reservation' use case. Verbs in Fig. 5 are used for specifying the customer and the clerk tasks, and the system responses. Nouns in Fig. 5 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. 5 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.

3.3. Conceptual object model

In ADOI a conceptual object model is similar to a normal domain model in form. The conceptual object model 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. Conceptual objects should be described in terms meaningful to the users and delivered in their entirety.

Fig. 6 shows a class diagram for the 'make reservation' use case. The conceptual model contains classes that are significant from a user's perspective of the task. 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.

Comparing the conceptual object model in Fig. 6 with the domain object model in Fig. 7, one can find that they are different to a great extent. Although some classes may have the same names, the attributes of them are usually different. For example, the 'reservation' class in fig. 6 specifies information about the current customer, but the 'reservation' class in fig. 7 maintains a historical list of all customers who have made reservations.

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 things in the domain by reorganizing information provided by these things. As a result, *users do not want the raw information, but rather they need the information to be restructured and summarized* [28]. However, it is difficult to discover the information for the restructuring and reorganization using an inside-out perspective, as proposed by [6]. 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, and to prevent domain knowledge from being leaked into the UI of the system.

3.4. Domain object model

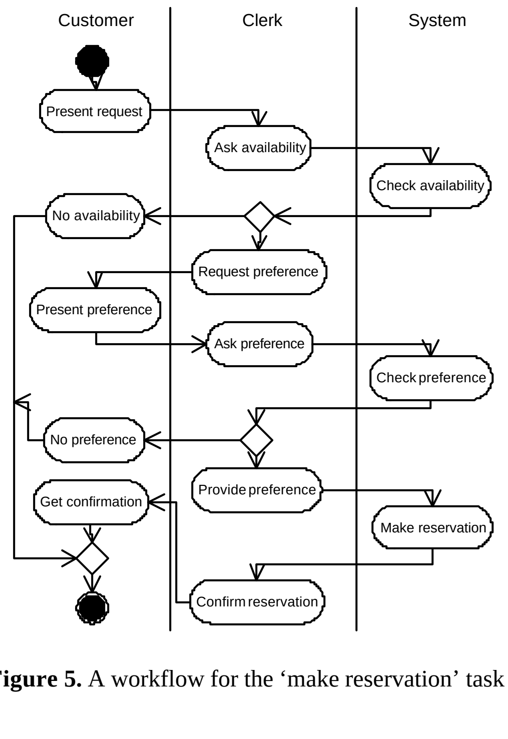

Figure 5. A workflow for the 'make reservation' task

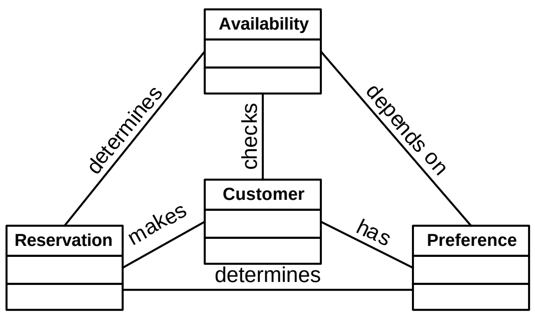

Figure 6. A conceptual object model for the 'make reservation' task

A domain model represents the understanding of domain-imposed requirements from a developer'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. 7 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* [7, 12, 17] 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 meanings or goals of user tasks, that is, the specified tasks for the envisioned system in the use cases have to map to corresponding domain features.

ADOI attempts to specify the relationships in a declarative way by a contextual link model. A contextual link maps a system task defined on the conceptual objects to a collection of semantic operations defined on the domain objects. The role of contextual links can be

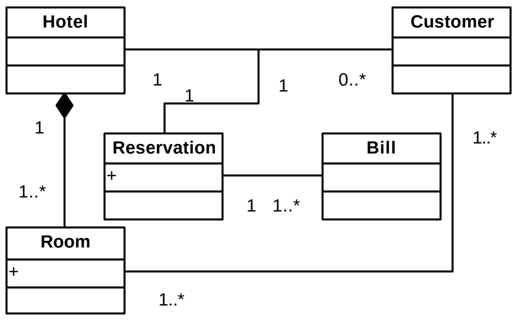

Figure 7. A domain object model for the example application (adopted from [25])

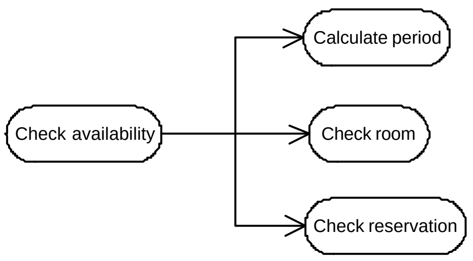

Figure 8. A contextual link model for the 'Check availability'

two-fold:

- Tracing domain objects that are relevant to tasks of the system
- 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. 8 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

More than a bridge between requirements and implementation, the role of architectural model for contextual development is to act as a means to deliver knowledge about problem-solving goals to the system design, as we mentioned earlier. Up to now we have conceptually modelled the knowledge both from a user's perspective and a developer's perspective. In the remainder of this section, we architect a system's structure in terms of the knowledge.

In ADOI, a component is modelled by the assumption over its functionality, which realizes a piece of the knowledge in declarative way. This means that the component is denoted with respect to what can be done with it rather than to how the component can be implemented.

Fig. 9 presents the assumed knowledge space realized by the ADOI architectural model. The space is composed of two subspaces:

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

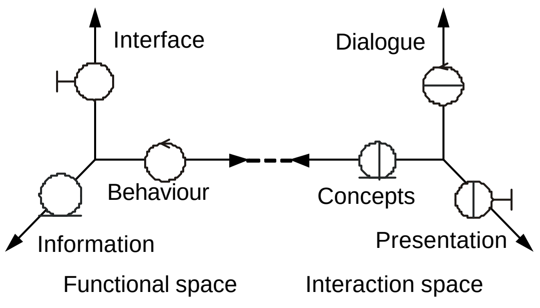

Figure 9. The ADOI knowledge space

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

The dimensions in the space are independent of each other. But there is also one shared dimension between the two subspaces. For example, the behaviour dimension is not depended on the dialogue dimension. 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 [25], 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 in 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.

This knowledge space is a contextual extension to original UML architectural framework specified by the UML analysis profile [19]. In fact, the UML profile only contains the functional space in the Fig. 9, in which the interface dimension is used to cope with the knowledge about both non-human and human agents, reflecting a system-centred perspective.

As a result, the components defined in ADOI architectural model are extended from UML analysis profile. ADOI architectural model defines six stereotypical classes:

- «Task-boundary» classes. A task-boundary class is assigned to or realizes 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 [27] 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 (conceptual objects), and that request information from the user, rather than *abstractions of physical user-interface components* [25].
- «Task-control» classes. A task-control class is assigned to or realizes knowledge about user tasks as multiple mapping relations with respect to functional respects. Task-control classes often cope with complex task-control logic related to use cases. They 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.
- «Task-entity» classes. A task-entity class is assigned to or realizes 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. They often encapsulate information matching user's mental representation and, therefore, they isolate change in data structure of the system and the user interface.
- «System-control» classes. A system-control class is assigned to or realizes knowledge about the contextual relationships between user tasks and system tasks. 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.
- «System-entity» classes. A system-entity class is assigned to or realizes 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 and often persistent. Their role is much similar to the 'entity class' in the UML analysis profile [19].
- «System-interface» classes. A system-interface class is assigned to or realizes knowledge about the relationships between the system and its environment. Their role is much similar to the 'interface-class' in the UML analysis profile [19].

Fig. 10 demonstrates an artefact of architectural modelling for the 'make reservation' use case in terms of knowledge in the conceptual models built for the example application. Objects in Fig. 10 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

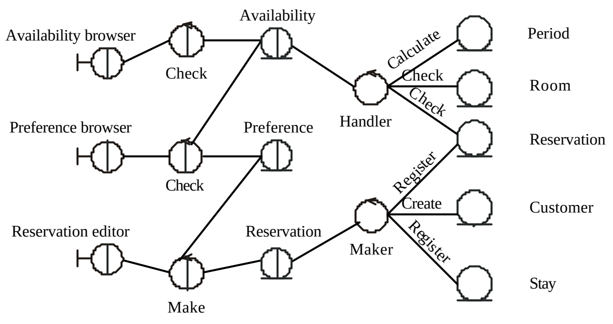

Figure 10. An analysis model for the 'make reservation' use case

'Availability' object is assigned to the functionality to present the 'Availability'.

Task-entity objects in Fig. 10, in fact, act as the mediator between the UI and the FC. They are used not only to map users' concepts between actual users and the UI, but also to implement 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. 10 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.

Different design methods are assumed for the purpose of implementation. For example, a task-boundary object represents a smaller aggregation of tasks. In order to deliver a design for the presentation and dialogue control of interaction, the visual design of task-boundary objects can be reified using a hierarchical task model.

4. CONCLUSIONS

User-centred design requires generic representation of usage for the system design rather than problem-solving specification for the component design. In order to achieve this point conceptualization performed for a particular design has to be generalized for the understanding and representation of usage of the system. The generalization is provided with the integration of declarative models, which allows an integrated process of development with a generic, open architecture. Models in ADOI have been built in this way. Therefore, ADOI overcomes some limitations of current approaches that suffer from problems of level-of-abstraction. Static and explicit concepts defined in those models allow a declarative specification for the usage, and the integration of different perspectives through declarative 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.

The further development of ADOI includes an application ontology that integrates all concepts in the models with a taxonomic structure and concerns relations between the concepts in the models and between the models with axiom. The ontology increases a shared understanding between people and reuse of conceptualization for similar applications in the same domain on the one hand. On the other hand, the ontology drives automatic representation of system architecture because of its formal, explicit specification.

REFERENCES

1. Maguire, M.: Methods to support human-centred design. Int. J. Human-Computer Studies, 55 (2001), 587-634

2. Lamsweerde, A.: Requirements engineering in the year 00: A research perspective. In the proceedings of ACM ICSE 2000, (2000), 5-19

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

5. Garlan, D.: Software architecture: A roadmap. In: The proceedings of ACM future of software engineering (2000), 93-101

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

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

8. 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)

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

11. Juristo, N., Moreno A. M.: Introductory paper: reflections on conceptual modelling, Data & Knowledge Engineering 33 (2000)

12. Studer, M. Benjamins, V. R., and Fensel Dieter: Knowledge Engineering: principles and methods. Data & Knowledge Engineering 25 (1998) 161- 197

13. Paterno, F.: Model-based design and evaluation of interactive application, Springer Verlag, (1999)

14. Puerta, A. and Einsenstein, J.: Towards a general computational framework for model-based interface development systems, ACM CHI'97, (1997)
15. Welie, M. *et al*: An ontology for task world models, proceedings of DSV-IS'98, Spriger Verlag, (1998) 57-70
16. Stary, C.: Contextual prototyping of user interfaces, proceedings of ACM Dis'00, (2000) 388-395
17. Gruber, T.R.: A translation approach to portable ontology specification. Knowledge Acquisition, 6(2): (1993) 199-221
18. Plihon, V. *et al*: A reuse-oriented approach for the construction of scenario based method, Proceedings of ICSP'98, (1998) 14-17
19. G. Booch, et al. "The Unified Modelling Language User Guide", *Reading, MA: Addison-Wesley*, 1999.
20. I. Jacobson, *et al*: Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, (1992)
21. Hsia, P. *et al*: Formal approach to scenario analysis, IEEE Software, 3 (1994) 33-41
22. Glinz, M: An integrated formal model of scenarios based on statecharts, Proceedings of ESEC'95, (1995) 254-271
23. Constantine, L.: Use cases for essential modeling user interfaces, ACM interaction, 4 (1995) 34-46
24. Robert, D. *et al*: Designing for the user with OVID, Macmillan, Indianapolis, Ind. (1998)
25. Nunes, N. and Cunha, J. F.: Wisdom: A software engineering method for small software development companies, IEEE Software, September/October, (2000) 113-119
26. Collins-Cope, M.: The RSI approach to use case analysis, Proceedings of TOOLS 29, (1999)
27. Norman, D.: Cognitive engineering, In: Norman, D. and Draper, S (eds.): User centered system design, Lawrence Erlbaum Associates, Hillsdale, NJ. (1986) 31-61
28. Szekely, P.: Retrospective and challenges for model-based interface development. In the proceedings of CADUI'96, (1996)