

# Representing Functional Requirements and User-System Interactions

**B. Chandrasekaran**

Laboratory for AI Research  
The Ohio State University  
591 Dreese Laboratories  
Columbus, OH 43210  
Email: chandra@cis.ohio-state.edu

**Hermann Kaindl**

Siemens AG Österreich, PSE  
Geusaugasse 17, A - 1030 Vienna Austria  
Email: kaih@siemens.co.at

## Abstract

Specifying the requirements for a new system to be built is a sufficiently important issue in systems engineering that it has become a research area of its own called *Requirements Engineering*. Related to this issue, designing and specifying the interactions of potential users with a system is an important problem in *Human-Computer Interaction*. In this paper, we apply *Functional Representation* (FR) to model functional requirements and user-system interaction, in the process clarifying their mutual relationship.

## 1. Introduction

It is widely accepted that a clear set of requirements facilitates system design — whether it is a software, hardware or a hybrid system. Requirements specification includes precise description of needed functionalities and required interactions between the user and the system, as well as so-called non-functional requirements (constraining the development process and the developed system). The more precisely and unambiguously these requirements are specified, the better off is everyone involved in the whole process: the customer, the system designers and implementers, and users. The required precision and lack of ambiguity can only be achieved if we have a clear understanding of the kind of things that need to be stated as part of the requirements — a clear identification of what has been called the *ontology* of the situation — and support the ontology by means of a formal representation vocabulary.

In this paper we focus on functional requirements and specification of needed interactions between the user and the system. Such interactions have been a subject of discussion in the literature on software engineering and the design of interactive systems (see for example, [1-4]). We will adapt a representation from a body of work known as Functional Representation (FR) (for a review of this work, see [5]) for specifying such interactions. We build on the work on requirements specification and task modeling using functional ideas reported in [6]. In a recent paper on applying functional representation to software reuse and design [7], requirements of some specified functional prototype from other functional prototypes are specified in its implementation. In contrast, we describe requirements of some user for a complete system to be built.

The outline of our argument in this paper is as follows. We discuss some desiderata for a representational framework for requirements. We discuss a definition of function and its relation to the purposes of a user. Together, these give us some terms for representing functions. The definition that we provide introduces the need for specifying how an artifact is to be used — the way a user is to interact with the device — as an intrinsic part of the task of design. We term this part of design *interaction design*. As interaction design proceeds, the interactions needed can be articulated to varying degrees of concreteness. A particularly common representation of such interactions is through *scenarios*, which capture the series of interactions between the user and the system needed for the function to be achieved. We show that such scenarios can be represented in a manner similar to

the so-called *causal process representations* in the FR literature. We motivate our discussion by using as a concrete example the Automated Teller Machine (ATM).

## 2. Desiderata for a Framework for Functional Requirements

We use the shorthand FIRQ to stand for functional and interaction requirements. We believe that an FIRQ framework should deal with the following issues to some degree.

1. *Specifying functions.* FIRQ should of course allow the specification of desired functions. In case where it is appropriate, it should also allow situations to be avoided, prevented, etc.
2. *Specifying interactions.* In the design of interactive systems, the customer would like to specify, at the design stage, that the intended functions are to come about as a result of certain interactions between the device or system and its user. FIRQ should support the specification of such interactions.
3. *Should not demand information not likely to be available at design time, but should allow representation of information that is available.* FIRQ are typically given before the design of internal structures and their connections (though a certain amount of it might evolve as a result of interaction between design and requirements modification). This means that FIRQ should not demand knowledge of the system — say its structure — that is not available at the time of requirements specification.
4. *Should allow elaboration and refinement of requirements as interaction design proceeds.* By the same token, FIRQ should allow specification of changing requirements as design proceeds and commitments are being made. As interaction design is performed, a more detailed set of requirements emerges. Thus a framework for FIRQ should ideally support requirements evolution during interaction design.

## 3. What is a Function?

In much of the work on representing functions, including in the work on FR, function is treated as a property of the object or device, often as some abstraction of a selected behavior of the device. As an example, the function `buzz` of the device `buzzer` might be defined

as follows: “When the switch is pressed by a user, a sound is produced in its clapper.” Note that, in this description, the switch and the clapper are parts of the buzzer, and the behavior of interest is described in terms of the states of these components or ports of the device. This definition certainly captures certain things we want from the definition of a function: that it expresses an intention of a designer or a user, that it is an abstraction of behavior and so on. However, the function cannot be defined if we do not have the device in the first place. Imagine that the buzzer has not been designed yet and a customer is looking for a device to do what the buzzer helps to accomplish. The customer — we will imagine her to be the user of the device — has a purpose in mind which she would like the device to accomplish or help her accomplish. How is this purpose to be represented?

Clearly, it cannot use aspects of the device not yet designed. One of us has argued, in a recent proposal on the definition of function [8], that a *function or role of an object is an effect it has on its environment*. The function defined in this way is a dual to the purpose of a user. The user intends — has the purpose to cause — a certain effect in her world, and if an object or a device can create the effect, then she may attribute the effect as a function of the object.

Let us assume an environment consisting of some objects. The objects may be specified abstractly and incompletely, as long the state variables of interest to us in modeling are available. (We only consider *dynamic functions* here, i.e., functions defined in terms of state variables. There are also what one might call *static functions*, such as the seating function of a chair or the light-passing function of a window, that are defined in terms of objects’ static properties. Such functions are discussed in [8], but we do not consider them further here.)

**Functions and Purposes.** A distinguished function or role is the occurrence of certain events or effects of interest in the environment. Let us say *F* stands for such an identified function or role. Intentional agents often have *purposes* to cause certain events or effects in the environment. If agents have a purpose to cause effect *F* in the environment, and, in order to achieve this purpose, if they use a certain object that causes *F*, then they may say that the object has the function *F*. Suppose a theorist wishes to explain a certain effect *F* in some domain. If she believes that some object *O* causes the effect *F*, she may say that *O* has the role *F* in the domain. All of these concepts use as their central element the idea of a distinguished effect of interest.

**Distinguished Effects (or Functional Predicates) of Interest.** The most general version of these is given by a

set of {conditions, effects} where both conditions and effects are specified as predicates or temporal sequences of predicates defined over environmental state variables. The functional predicates or effects are defined purely in terms of environmental variables. They make no reference to the properties of any object that might be introduced into the environment to cause the effects.

**Examples.** The *buzz* functional predicate. A selected part of the environment has a buzzing sound in it.

The *sawtooth* functional predicate. The voltage between two given terminals in the environment to rise over interval *T* from 0 to *V*, then instantaneously drop to zero, and this pattern to be repeated.

**Function of an Object.** Given a functional effect, how do we relate it to the function of an object? Since the functional predicates and conditions may not make any reference to any part of the structure of the object, we need to describe a *mode of deployment* of the object to make the link between an object and defined effect of interest.

**Mode of Deployment.** Given an object *O*, a mode of deployment specifies:

1. How *O* is to be connected to its environment, i.e., how it is to be configured such that the environment can effect certain selected properties of *O* and *O* can effect selected properties of the environment. A typical way to specify this is to define *ports* of different types for *O* as well as the objects in the environment and describe which ports of *O* are connected to which ports of the environment.

2. Required external causes on the object, or, more generally, required sequences of interactions between *O* and external objects. In the case of a device that is to be used by an external user (as opposed to a device to be connected to other devices to make a composite device), the external causes are specified in terms of actions by a user on *O*.

**Scenarios.** The sequence of interactions between the user and a system has been called a *scenario* in the literature on interactive systems and Software Engineering [1-4]. Kaindl [6] emphasized that these are *required* interactions, and so scenarios can be viewed as *behavioral requirements*. We call them *interaction requirements* in this paper, and in order to make the notion of a scenario less ambiguous we call it an *interaction scenario*.

**Ascribing a function to an object.** Given a function *F* and an object *O*, and a mode of deployment, *M*, we can say that *F* is a function of *O*, if there is a mode of deployment *M* such that *O* under *M* causes the effects specified in *F* under associated conditions.

**Intended Function and User Purpose.** Function as defined above is neutral with respect to whether the effect on the environment is intended, as in the domain of devices, something undesired, as in the effect that a malfunctioning device might have on the environment, or simply a description of a fact, as in scientific descriptions where talk of intentions of nature are to be avoided. The more neutral term such as “role” is used to describe the latter. We elaborate here our earlier discussion on the relation between agents’ purposes, roles of objects and functions of devices.

*User purpose:* a user intends or desires a certain effect in the world under certain conditions. These effects are described using the {conditions, effects} formalism described earlier. Let us say that the user intends the effect *F*.

*Designer task:* The task of the designer is as follows. He is given *F* as part of the requirements and has to:

1. describe an object, i.e., a set of components from some agreed on repertoire of objects and their configuration, and
2. a mode of deployment of the object

such that under the described mode of deployment, the object causes the effects in *F*. Then a function, defined by *F*, can be attributed to the object.

A main point here is that what unites the user’s purpose, the designer’s task and the function of the object is the effect on the environment. The object causes those effects and thus has a function defined in terms of the effects. The user wants those effects, and hence looks for an object which has a function of causing those effects. The designer is tasked with making an object which has the function.

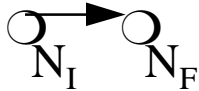
## 4. FIRQ Specification for ATM

Now we are ready to discuss requirement specification using the ATM example. Kaindl [6] links scenarios (in the sense of behavioral/interaction requirements) with functional requirements, and uses earlier FR work to define the underlying semantics. In this section, we develop this more precisely and elaborate on it. Interaction scenarios are one aspect of the “mode of deployment” that we talked about earlier.

**The Environment.** Consider an environment composed of a bank with customer account records, and (unspecified

number of) persons some of whom are bank customers (that is, they have bank accounts). We will denote a generic user by  $U$ , a generic customer by  $C$ , a customer's account by  $\#(C)$ , the balance in the account by  $B(\#(C))$ , and the cash that a user  $U$  has by  $c(U)$ .

**Effect or Functional Predicates for the ATM's Withdraw-cash( $\$w, \$L$ ) Function.** Let us say the bank officials would like a device one of whose functions is to let legitimate customers withdraw cash, up to a limit  $\$L$ , from their accounts. We define the functional predicates as in the following figure.



We define the effects of interest by defining an initial state  $N_i$  and a final state  $N_f$ , each with certain properties. We would like the device to cause the transition from  $N_i$  to  $N_f$ . Using the {conditions, effects} formulation,  $N_i$  defines the initial predicates and  $N_f$  the final predicates.

Functional (Effects) Predicates
$N_i$ is defined by the predicates: $c(U) = \$x, B(\#(U)) = \$y$
$N_f$ is defined by the predicates $c(U) = \$x + \$w, B(\#(U)) = \$y - \$w$

One purpose of the bank officials for the ATM can be called  $Withdraw\_cash(C, \$w, \$L)$ , i.e., they would like their customers to withdraw cash (within the withdrawal limit  $\$L$  set per withdrawal). The purpose is to cause the above predicates to be true under the following conditions:

Conditions
U's purpose is $(c(U) = \$x + \$w)$
U is a C
$\$w < \$y, \$L$

They would like a device which can cause  $N_f$  to be true, given  $N_i$  as the initial state and under the conditions above.

The function of the device ATM can also be called  $Withdraw\_Cash(C, \$w, \$L)$  and defined as the causing of the effects described under above conditions.

The purpose of a user  $C$  regarding an ATM can be described at several levels of description.

1.  $Get\_cash(\$w)$ , where  $\$x$  is his cash reserve at the initial state and  $\$x + \$w$  is the cash at the final state.
2.  $Withdraw\_cash(\$w)$ , where the cash reserves at the initial and final states are as in 1 above, but, additionally, imposes conditions on the balance in his account.

$Withdraw\_cash(\$w)$  is a special case of  $Get\_cash(\$w)$ . When the issue is to get some cash ( $Get\_cash(\$w)$ ), a bank customer may choose  $Withdraw\_cash(\$w)$  and might look for a device which will cause the corresponding  $N_f$  to become true<sup>1</sup>. In that case, the ATM might be a possible device, since its functional definition suggests that the ATM can cause  $N_f$  to become true. Since there are many other ways to realize  $Get\_cash(\$w)$  — borrow from someone, steal it, and so on — it is much more appropriate for the user to ascribe the  $Withdraw\_cash(C, \$w)$  function to the ATM than the more general function of  $Get\_cash(\$w)$ .

The main point of the above discussion is to illustrate the central role played by the functional predicates in defining the function abstractly and in relating the function to purposes of agents, users and designers alike. There are small differences in the way we defined the purposes of the bank officials and a user, e.g. — the officials might be more naturally interested than a user in limiting the amount of withdrawal to  $\$L$ . There are also differences in the way the function of the ATM and the purpose of a user are defined — the function is defined as allowing any customer to withdraw cash, while for a given customer  $C$ , his purpose is defined in terms of *his* being able to withdraw cash. These minor differences aside, the functional predicates described in the table are at the heart of descriptions of the purposes of the various intentional agents and the function of the ATM.

### Interaction Design Refinement

The function as specified above can be given to the designer as part of the requirements. Let us imagine that either the designer and/or the bank officials engage in some additional interaction design. The product of this

<sup>1</sup> How one matches a goal (in this case  $Get\_cash(\$w)$ ) to a device function ( $Withdraw\_cash(\$w)$ ) is an issue that recurs in the literature on reasoning about function. For example, Umeda, *et al* [9] discuss search for a component that can fulfill a given function. Liver [10] describes an algorithm for incrementally backing off of requirements until a matching function can be found. The issues related to matching are important, but not central to our main points, so we do not discuss this issue further in this paper.

design is going to be certain commitments about the way the device is to achieve its function. In particular, we wish to focus on specifying the required interactions between a user and the device.

**Scenarios Emerge from Design by Function Decomposition.** As mentioned earlier, the designer's task is to come up with a device and instructions for deploying it so that the function is achieved. Deploying a device involves specifying input and output ports and any user interactions needed at the ports to achieve the function. The function is defined as a set of predicates to be true at a final state (given another set of predicates is true at the initial state). These predicates can often suggest a top-level decomposition of the design task into subtasks. The point of interest to us here is that some of the subtasks may have a solution involving user interaction. Thus, as we move from a function definition which only focuses abstractly on the predicates intended to be true, to a series of refinements and decompositions into subtasks, an interaction scenario starts emerging as well.

In the ATM example, by looking at the function definition, we can identify a number of subtasks that the device has to perform:

1. Give U a means of expressing purpose, "Withdraw \$w."
2. Verify U is a C.
3. Verify  $\$w < \$L$  and  $< B(\#(C))$ .
4. Update  $B(\#(C))$  by subtracting  $\$w$  from it.
5. Dispense  $\$w$  if 2 and 3 above are satisfied.

We can get some information regarding subtask ordering by examining the preconditions of the subtasks: clearly subtask 1 has to precede subtask 3, and subtasks 2 and 3 have to precede subtasks 4 and 5.

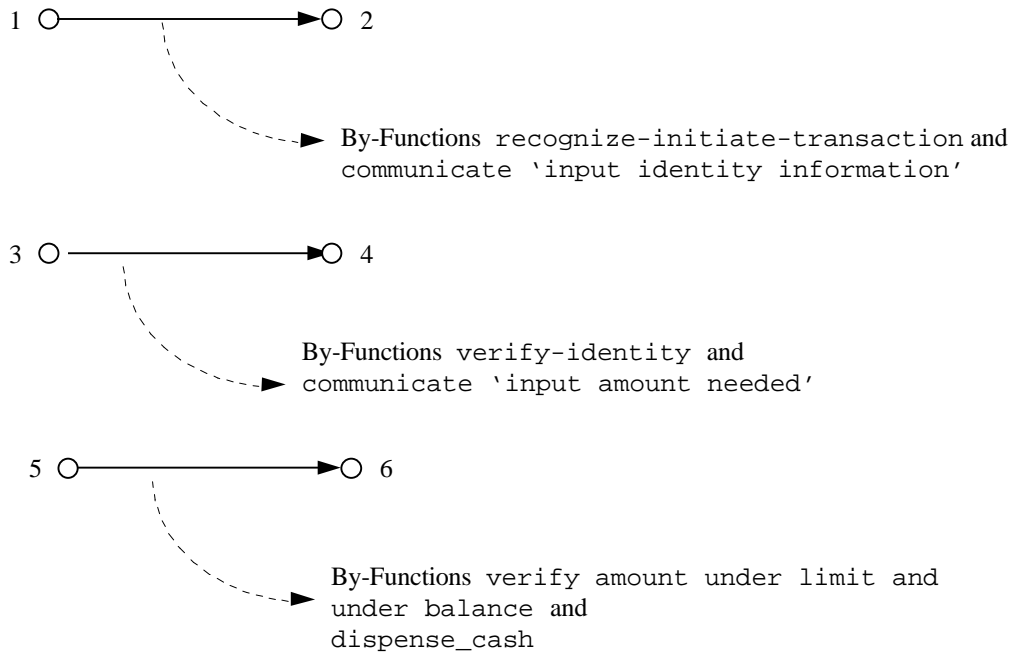
Let us just focus on the user-device interaction. Subtasks 3 and 4 do not require any interaction with the user — they call for interaction with the bank accounts database. Subtasks 1, 2 and 5 together determine the basis for the interaction scenario.

The scenario itself can be abstract [3] — making little commitment to the details of the device structure — or it can be concrete, involving commitments about ports, the places where user-device interaction takes place.

The design task decomposition above immediately suggests an *abstract scenario* of interaction:

1. The user initiates a withdrawal transaction
2. The ATM requests identification
3. The user provides identification
4. The ATM asks for the amount needed
5. The user communicates the amount needed
6. The ATM dispenses cash
7. The user takes the cash.

Each of the items in the sequence above is an *action*, either of the user or of the system. In this representation, we treat actions as the user being in certain states. Actions have effects on things acted upon just like any state of an entity that causally affects a state of another entity. Thus, the abstract scenario is a kind of causal process description. The transitions are one of two types: it either involves a device (ATM) function or a user function. Following the CPD representation, we can annotate the transitions corresponding to the ATM functions as follows.



The transitions 2 to 3, 4 to 5 and 6 to 7 involve user actions. We can, if we like, annotate them using By-Function  $\diamond$  of User,” but we have not done so in the above diagram because our focus is on the functional requirements of the ATM. If the ATM is embedded as a component in a larger system, where the user functions are performed by some other component of the larger system, then of course, it would be quite natural to encode the transitions using the By-Function annotation. In fact, this uniform way of treating user functions and component functions is one of the attractions of the CPD approach to encoding user interactions — users perform certain functions which then enable the device to go into certain states where they then perform other functions.

Each of the functions above can be defined using the functional definition framework described earlier. Some of the functions in the annotations of the transitions can be thought of functions of the components (yet to be designed) of the ATM while others can be attributed to the ATM as a whole. In the course of Requirements Engineering, it is not yet to be defined which components the ATM consists of. However, the ports can be defined where the effect of a function is to be achieved.

For example, consider `dispense_cash($w)`. Let  $P$  be a specified port of the ATM. The functional predicate that defines the function is:

Condition: Given \$0 at port  $P$   
 Effect: (ToMake) \$w at  $P$

The above abstract scenario, along with the functions that account for the transitions, can serve as part of the

requirements specification. The scenario can be much more concrete as well — if, as part of initial interaction design, additional commitments are made, and thus become part of the charge to the designers.

**More Concrete Scenarios.** Suppose it is decided that users will have a card with their account numbers, they will be assigned a password, and that (overall) the following ports will be available for interaction.

- $P_1$ , for user placing the card
- $P_2$ , for user to input password and desired amount
- $P_3$ , for the ATM to inform user of actions needed
- $P_4$ , for delivery of cash

The scenario can now be written as:

1. The user places card at  $P_1$
2. The ATM displays message at  $P_3$  : “Input password”
3. The user inputs password at  $P_2$
4. The ATM displays message at  $P_3$  : “Input the amount needed”
5. The user inputs the amount needed at  $P_2$
6. The ATM deposits cash at  $P_4$
7. The user takes the cash from  $P_4$ .

The transitions can be annotated as before, except that the functions can be much more specific about the ports at which certain functional predicates have to apply.

**Why the CPD Works for Representing the Interaction Scenario.** Why does the process description formalism developed for explaining how a device works prove useful for specifying user-system interactions? One can view the user plus the device as yet another “device” in which the user herself is a component causally interacting with another component. Thus, user’s action states are like the states of any other component. The interaction scenario simply becomes the causal process description for this larger device. While for the presentation above we had attributed the function `Withdraw_Cash(C, $w)` to the ATM device alone, actually the ATM and the user have to cooperate in order to achieve it. Also the user, as a component, has a certain number of functions to achieve, just like any other component such as the ATM. The interaction scenario then becomes a causal process description of how the user-ATM combination works. So, `Withdraw_cash(C, $w)` is really the function of the ATM plus the user. That is why, as we refine the design, the interaction scenario emerges from functional decomposition, by making explicit also the functions required from the user.

**How Such a Representation Can Be Used.** In earlier work of one of us (see Kaindl [6]), the primary use of functional representation was to define the underlying semantics of the new approach of linking interaction scenarios with functional requirements and purposes. The cleaner and elaborated representation described above should be even more useful in this respect.

When making use of one of the systems available for functional *reasoning*, our approach may also be useful for automated *analysis* of requirements. Requirements models represented in this way can be simulated in order to identify problems or validate the requirements.

## 5. Concluding Remarks

The contributions of this paper can be viewed from several perspectives. At the simplest level, it shows the use of a definition of function and the causal process description formalism of the FR framework to represent functional requirements for system design. The approach is as useful for software systems as it is for hardware or hybrid systems, since the terms used to describe functions equally apply to all types of systems. The function framework used here helps to see in a unified way how user purposes in using a device, designer intentions and functions of the device are related and arise from certain basic functional predicates defined in terms of environmental variables.

At another level, the work presented can be viewed as a formalism for representing user-system interactions, a

topic of substantial interest to the community concerned with the design of interactive systems. Again, there are a number of unifications: the same representational framework that is used for causal process representations in FR is used to represent user-system interactions. The functional annotations for transitions in CPD set a number of design subtasks for the designer.

Still another dimension of interest is the relation between interaction design commitments and refinement of functional requirements. In summary, this paper shows how modeling in the sense of functional representation and reasoning can be usefully applied to Requirements Engineering and Human-Computer Interaction.

## Acknowledgments

B. Chandrasekaran’s research was supported by ARPA, order no. A714, and monitored by USAF Rome Laboratories, contract F30602-93-C-0243. The authors acknowledge with thanks comments by Dean Allemang that helped improve the paper.

## References

- [1] Carroll, J. M. ed. *Scenario-Based Design*. New York, NY: John Wiley & Sons, 1995.
- [2] Carroll, J. M.; Mack, R. L.; Robertson, S. P.; and Rosson, M. B. “Bindings scenarios to objects of use,” *International Journal of Human-Computer Studies*, vol. 41, pp. 243-276, 1994.
- [3] Constantine, L. “Essential Modeling: Use Cases for User Interfaces,” *ACM Interactions*, vol. 2, pp. 34-46, 1995.
- [4] Potts, C.; Takahashi, K.; and Anton, A. I. “Inquiry-based requirements analysis,” *IEEE Software*, vol. 11, pp. 21-32, 1994.
- [5] Chandrasekaran, B. “Functional representation and causal processes,” in *Advances in Computers*, vol. 38, M. C. Yovits, Ed.: Academic Press, 1994, pp. 73-143.
- [6] Kaindl, H. “An integration of scenarios with their purposes in task modeling,” in *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, Methods & Techniques (DIS ‘95)*, pp. 227-235, Ann Arbor, MI, 1995, ACM.
- [7] Liver, B., and Allemang, D. T. “A Functional Representation for Software Reuse and Design,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 5, pp. 227-269, 1995.
- [8] Chandrasekaran, B. “An explication of function,” The Ohio State University, Laboratory for AI Research, Columbus, OH, Draft, 1996.
- [9] Umeda, Y.; Tomiyama, T.; and Yoshikawa, H. A design methodology for a self-maintenance machine

based on functional redundancy, in *Design Theory and Methodology DTM 92*, D. L. Taylor and L. A. Stauffer, Ed., American Society of Mechanical Engineers, 1992, pp. 317-324.

[10] Liver, B. Working-around faulty communication procedures using functional models, in *Working Notes on the AAAI-93 Workshop on Reasoning about Function*, 1993, pp.95-101.