# Toward a Domain Description with CafeOBJ

Yasuhito Arimoto†, Masaki Nakamura†, Kokichi Futatsugi†

†Graduate School of Information Science,
Japan Advanced Institute of Science & Technology
arimotoy, masaki-n, kokichi@jaist.ac.jp

In our work, we try to describe a domain description with CafeOBJ. A domain description means a document which describes observable phenomena of the domain. It also means the process of of domain capture, analysis and synhesis, and the document which result from that process[3]. We formalise the domain description in [2] in CafeOBJ. CafeOBJ is a formal specification language. By formalisation of domain description, we can avoid the ambiguities and inconsistency of the description, and can prove propertiess in a domain precisely.

## 1 Introduction

In [3], the triptych dogma is introduced as follows.

- Before software can be designed, programmed, coded, its requirements must first be reasonably well understood.
- Before requirements can be expressed properly, the domain of the application must first be reasonably well understood.

From a description of the application domain, we can construct the prescription of the requirements, and from the prescription of the requirements, we can construct the specifications of software.

In our work, we focus on the understanding the domain of the application, which is the first thing to do for the software development, according to the trptych dogma.

CafeOBJ[1, 4, 5] is a formal specification language. It is a direct successor of OBJ and it inherits all its features. We try to describe domain descriptions with CafeOBJ. By formalise the domain descriptions in CafeOBJ, we believe we can describe the domain descriptions more precisely, and we can get more understanding of the domain from the process of descriptions. The specifications written in CafeOBJ is executable, and it helps us to prove the theories hold in the domain. This executability of CafeOBJ also help us to understand the domain.

In section 2, we introduce domain descriptions defined in [3] and the formal specification CafeOBJ. And in section 3, we introduce a domain description of hospitals in CafeOBJ. section 4 is the conclusion of this work as far as we have done. In section 6, we consider about another way of describing the domain descriptions in CafeOBJ.

## 2 Preliminary

### 2.1 Domain Descriptions

According to [3], domain descriptions and terms related to them can be characterised as follows.

**Characterisation of Application Domain**

By an applycation domain we shall understand anything to which computing may be applied.

**Characterisation of Domain**

By a domain, we mean an application domain.

**Characterisation of Domain Descriptions**

1. A domain description is a something which describes observable phenomena of the domain: entities, functions over these, events, and behaviours.
2. A domain description is also the process of domain capture, analysis and synthesis, and the document which results from that process.

Entities, functions, events, behaviours, and terms related to them are also characterised in [3] as follows.

**Characterisation of Entities**

By an entity we shall loosely understand some-

thing fixed, immobileor static. Although that thing may muve, after it has moved it is essentially the same thing, an entity.

**Characterisation of Atomic Entities**

By an atomic entity we shall understand an entity which cannot be understood as composed from other entities.

**Characterisation of Composite Entities**

By a composite entity $e$ we shall understand an entity which can best be understood as composed from other entities, called the subentities, $e_1, e_2, ..., e_n$, of eneity $e$.

**Characterisation of Subentities**

By a subentity, we shall understand an entity which is a component of another entity.

**Characterisation of Values**

By a value $v_e$ of an entity we shall loosely understand the following: If the entity is an atomic entity, then the entire set of identified attributes, $a_{1_e}, a_{2_e}, ..., a_{n_e}$, of the entity. If the entity is a composite entity, suppose subentities are $e_1, e_2, ..., e_m$, then there are three parts to the entity value: how it is composed — its mereology $m$, the entire set of identified attributese, $a_{1_e}, a_{2_e}, ..., a_{n_e}$, of the entity, and (inductively) the identified values, $v_{e_1}, v_{e_2}, .., v_{e_m}$, of respective subentities $(e_1, e_2, ..., e_m)$.

**Characterisation of Attributes**

By an attribute of an entity we shall loosely understand a quality which cannot separated from the entity.

**Characterisation of Entity Mereology**

By mereology we understand a theory of parthood relations. That is, of the relations of part to whole and the relations of part to whithin whole.

**Characterisation of Functions**

By a function we shall loosely understand something, a mathematical quantity, which when applied to something, called argument of the function, yields something, called result of the function for that argument. If the function is applied to something which is not a proper argument of the function, then the totally undefined result, called chaos, is yielded.

**Characterisation of States**

By a state we shall loosely understand a collection of one or more entities whose value may change.

**Characterisation of Actions**

By an action we shall loosely understand somethig which changes a state.

**Characterisation of Events**

By an event we shall loosely understand the occurence of something that may either trigger an action, or is triggered by an action, or alter the course of a behaviour, or a combination of these.

**Characterisation of Behaviours**

By a behaviour we shall loosely understand a sequence of actions and events.

### 2.2 CafeOBJ

The basic building blocks of CafeOBJ[1, 4, 5] are modules and the module mainly consists of two parts. One is the signatures, and the another is the axioms. Signatures are formed by a set of sorts and operations on the set of sorts. Axioms shows how the operators function by using equations. Here is an example of the CafeOBJ specification of the strings of natural numbers.

```
mod! STRG-NAT {
  pr (NAT)
  [ Nat < Strg ]

  op nil : -> Strg
  op (_._) : Strg Strg -> Strg { assoc }

  var S : Strg

  eq (nil . S) = S .
  eq (S . nil) = S .
```

`mod!` is the tight semantic notation. For loose semantic, `mod*` is used. `NAT` is the name of the module. Signatures consists of importing modules, sort declaration, and operator declaration. `pr` (*module-name*) is for importing module *module-name*. In this specification, the module which specifies the data type natural numbers is imported. Sorts are declared by the notation `[ ]`. `[ Nat < Strg ]` shows the sorts `Nat` and `Strg` are declared, and `Nat` is the sub sorts of `Strg`. Operations declarations begins with `op`. Operations declarations consist of the name of the operation, which can be mix-fix syntax with showing the position of the arguments by "_", the arity of the operation, and the sort of

the operation. An arity of the operation may consists of an empty string (like in the case of nil), only one sort, and several sorts (like "`Strg Strg`" in the case of `(_._)`. Note that these sorts of the arity may be different). `{ assoc }` shows the operation satisfies associativety. Variables declarations begins with `var` or `vars`. `vars` are for declaring more than one variable for a sort. Axioms are declared by using equations, and it begins with `eq`.

# 3 Example of a Domain Description in CafeOBJ

In this section, we introduce a CafeOBJ specification of a domain of hospitals which is the description of the entities and the functions on the domain of the hospitals. Events can be written as comments in the specification, because they are just what happen after or before the actions. Events themselves do not change any states of entities. Behaviours are sequences of actions and events. We think that we do not have to formalise these.

Figure 1 shows how hospitals are constructed. We postulate entities of the domain of the hospital are the hospital, the ward, the operating room, the pharmacy, the administrational staff, the patients, the medical staff, the medical staff station, the patients medical records, the medicine box, the medicine, the medical machine, the medical tool, the bedroom, the bed, and the consultation room. The hospital consists of the wards, the operating rooms, the pharmacy, the administrational staff, the patients, and the medical staff. The wards consists of the medical staff stations, the bedrooms, and the consultation rooms, and the other entities are constructed as figure 1.

Functions are admit, interview, plan analysis, analyse, diagnose, plan teatments, treat, transfer, and release. When citizenss come to a hospital, they are admitted and become patients and the patients medical records for them are created. After the admitted, patients get interview. Based on the result the interview, the medical staff make an analysis plan. After the analysis, medical staff diagnose the patients, and make a treatment plan for the patients based on the result of the diagnosis and treat them. If the patients get another symptom, then the patients are transferred to another ward. If the patients get curred, then the medical staff release the patients. The the procedure of the treatment for a patient obeys the patients medical record for him or her.

## 3.1 Entities of the Domain of Hospitals

Before showing how to describe the domain of hospital, we define the datatype of sets. This module SET is a parameterized module, and it is useful to describe the set of something, by giving an argument.

```
mod* SET (X :: TRIV) {
  pr ( EQL )
  pr ( NAT )
  [ Elt Empty NeSet < Set ]

  op none : -> Empty
  op _ _ : Set Set -> NeSet
                    { assoc comm idem idr: none}
  op _ in _ : Elt Set -> Bool
  op card : Set -> Nat
  op del : Elt Set -> Set

  vars E E' : Elt
  vars S S' : Set

  eq E in none = false .
  eq E in (E' S) = if (E = E') then true
                   else (E in S) fi .
  eq del (E, none) = none .
  eq del (E, E') = if (E = E') then none
                   else E' fi .
  eq del (E, (E' S)) = if (E = E') then del (E, S)
                       else (E' del(E, S)) fi .
  eq card (none) = 0 .
  eq card (E) = 1 .
  eq card (E S) = 1 + card (del (E, S)) .
}
```

In this case, the sort `Elt` is a parameter, that is, `Elt` can be replaced by another sort.

### 3.1.1 Atomic Entities and Composite Entities

Atomic entities have attributes and composite entities have attributes and subentities. To describe entities in CafeOBJ, we define entities as a pair of a set of attributes and a set of sets of subentities. Therefore, entities have the form as following.

Entity = (Attribute set) × ((Entity set)set)

where (Attribute set) is a set of attributes, and ((Entity set) set) is a set of sets of entities. For

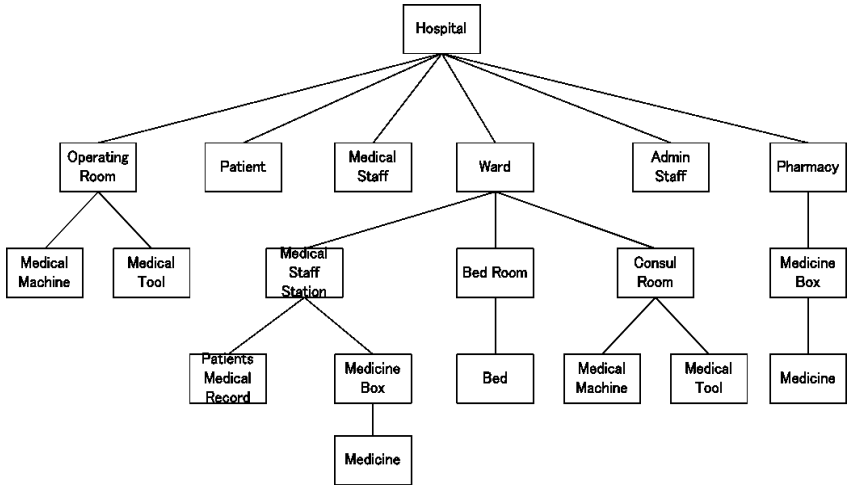


図 1: “The Hospital”

atomic entities, ((Entity set) set) is the empty set. Figure 2, 3 shows the tree structure of the entities.

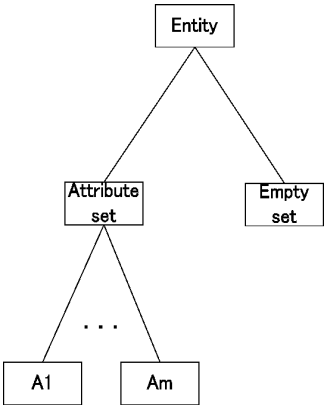


図 2: “Atomic entity”

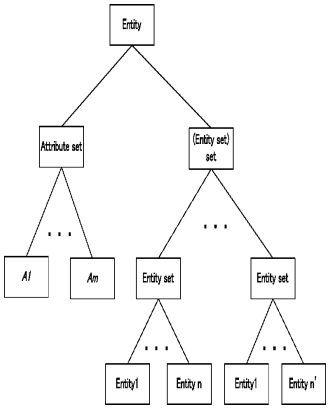


図 3: “Composite entity”

We define each entity in a module respectively, except sort declarations of attributes. The sorts for attributes are declared in a module, because some entities have the same attributes as others. If these are declared in defferent modules, they are regarded as different sort although they should be the same sorts. Assume that the module in which sorts of attributes are declared is named `ALLATT`, and the module for the atomic entity patient can be described in CafeOBJ as follows.

```
-- Patient
mod! PATIENT principal-sort Pa{
  pr ( ALLATT )
  [ PAttri PSub Pa ]

-- Attributes set.
  op pAtt : Name WID Rnm BID PMRID Loc NmSet
            -> PAttri

-- A patient is an atomic entity.
  op pSub : -> PSub

-- Constructor of the patient.
  op pa : PAttri PSub -> Pa

}

mod! PASET {
  pr ( SET ( PATIENT )
           * { sort Empty -> PaEmpty,
               sort NeSet -> NePaSet,
               sort Set -> PaSet,
               op none -> nonePa } )
}
```

The entity patient is represented as the sort `Pa`. `Name`, `WID`, `Rnm`, `BID`, `PMRID`, `Loc`, and `NmSet` are the

attributes of Pa and they are declared in the module ALLENT. Name is the name of the patient, WID : the ID of the ward where the patient is taken care of, Rnm : the room number of the bedroom where the patient is staying, BID : the ID of the bed which the patient is using, PMRID : the ID of the patients medical record for the patient, Loc : the location where the patients is in the hospital, and NmSet : the set of names of medical staff who take care of the patient.

PAttri is the set of attributes of Pa, and pAtt is the constructor of PAttri. PSub is the set of sets of subentities of Pa and pSub is the constructor of PSub. Since Pa is an atomic entity, it has no subentity. pa is the constructor of Pa.

To specify the set of patients, we reuse the module SET. By substitute the sort Pa in the module PATIENT, we can spesify the set of Pa. Since Pa is declared as a principal sort of the module PATIENT, CafeOBJ system can get to know Elt should be replaced by pa by the notation SET ( PATIENT).

The composite entity hospital is described in CafeOBJ as follows.

```
-- Hospital
mod! HOSPITAL {
  pr ( WSET  )
  pr ( ORSET )
  pr ( PHSET )
  pr ( PASET )
  pr ( MSSET )
  pr ( ASSET )
  pr ( ALLATT )
  [ HAttri HSub Hos ]
-- Attributes are the name of the hospital
-- and the location.
  op hAtt : Name Loc -> HAttri

-- Subentitis of a hospital are the ward,
-- the oparating room, the pharmacy,
-- the hospital central staff, and the patient.
  op hSub : WSet ORSet PhSet ASSet PaSet MSSet
            -> HSub

-- Constructor of the hospital.
  op hos : HAttri HSub -> Hos

-- Check mereology
  op mere : Hos -> Bool

  var HA : HAttri
  var WS : WSet
  var ORS : ORSet
  var PhS : PhSet
  var ASS : ASSet
  var PaS : PaSet
  var MESS : MSSet
```

```
-- Mereology—   eq mere(hos(HA,
              hSub (WS, ORS, PhS, ASS, PaS, MESS)))
       = if ((card(WS) > 0) and
             (card(ORS) > 0) and
             (card(PhS) = 1) and
             (card(ASS) > 0) and
             (card(PaS) >= 0) and
             (card(MESS) > 0)) then true
         else false fi .
}
```

The entity hospital is represented by the sort Hos. HAttri is the set of attributes of Hos and hAtt is the constructor of it. HSub is the set of sets of subentities of Hos and hSub is the constructor of it. hos is the constructor of Hos. mereis the predicate to check whether Hos satisfies the mereology of hospitals. The mereology of the hospitals is that it consists of one or more wards, one or more operating rooms, one pharmacy, one or more administrational staff, zero or more patients, and one or more medical staff.

### 3.2 Functions of the Domain of Hospitals

In this section, we show how functions are specified in CafeOBJ. To define actions, we firstly define the observers for the attrubutes of entities in the module ATTOBS. By using these observers, we specify functions over the domain of hospitals.

We specify functions in the module FUNCTION and following shows the definition of the function admit which is a part of the module FUNCTION.

```
pr ( CITIZEN )
pr ( HOSPITAL )
pr ( ATTOBS )

  op admit : Cit WID Hos -> Hos
  op addP2H : Pa Hos -> Hos
  op creatPMR : Name PMRID WID Hos -> Hos
  op pmrid : Cit Hos -> PMRID

  var C : Cit
  var Wid : WID
  var H : Hos
```

```
-- admit
  eq admit(C, Wid, H) =
       creatPMR (cname (C),
                 pmrid (C, H),
                 Wid,
                 addP2H (pa (pAtt (cname(C),
                                   Wid,
                                   undefR,
                                   undefB,
                                   undefP,
                                   L,
                                   noneNm),
                             pSub),
                         H)) .
```

Arities of `admit` is `Cit` (the citizen), `WID` (the ID of the ward), and `Hos` (the hospital), and the sort of `admit` is `Hos`. `admit` is the function for admitting citizens to the hospital as patients. What this function actually do is add a new patient to the set of the patients which is in the hospital and creat a new patients medical record for the patient in the medical staff station of the ward whose ID is the same as the one given as an arity of `admit`. Operation `addP2H` is for former and `creatPMR` is for latter. `pmrid` is the constructor of `PMRID`. We do not care how hospitals the ID for the patients medical records are created, but we only cares it should be distinct. `cname` is the obsever for the name of citizens which is defined in the module `ATTOBS`.

## 4 Conclusion

For a domain description of hospitals in CafeOBJ, we regard entities as a pair of a set of attributes and a set of sets of subentities. In the example of a domain description of hospitals we showed, this modelisation worked. And based on the specification of entities, we are also be able to define functions for the domain.

## 5 Future Works

In this work, we do not reach to the step for the proving properties of domains. One instance of the propertes is the mereology between entities. Since we defined the predicate for each entities which checks whether the entity satisfy the mereology, we can define the predicate which checks not only mereology for the entities but also the mereology of subentities. In the future, we try to prove if the mereology of entities and subentities is satisfied after any functions on the domain is applied. We think that is one property of the domain which should be hold.

For the description in CafeOBJ introduced in this document, we specified the entities as data types which is denoted by visible sorts. As a result, the definition of functions get complicated and not easy to read. Another way of describing domain in CafeOBJ, we regard entities as state spaces which is denoted by hidden sorts. It may makes the specification of the domains easier to see.

## 6 Acknowledgement

The authors are grateful to all people who contributed to the discussion of domain descriptions and development of the CafeOBJ specification for domains, especially Prof. Dines Bjorner, Dr. Jianwen Xiang, and Miss Xiaoyi Chen.

### 参考文献

[1] CafeOBJ: CafeOBJ web page. `http://www.ldl.jaist.ac.jp/cafeobj/`

[2] Yasuhito Arimoto. A domain Description of “THE HOSPITAL”. Course report, JAIST, School of Information Science, 1-1, Asahidai, Nomi, Ishikawa, Japan 923-1292, Spring 2006.

[3] Dines Bjorner. *Software Engineering, Vol 3: Domains, Requirements and Software Design.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[4] Răzvan Diaconescu, Kokichi Futatsugi. *CafeOBJ Report.* AMAST Series in Computing, 6. World Scientific, Singapore.

[5] Ataru Nakagawa, Toshimi Sawada, Kokichi Futatsugi. *CafeOBJ User's Manual Ver.1.4.* `http://www.ldl.jaist.ac.jp/cafeobj/documents.hrml`