TECHNICAL REPORT

RT – ES 719/08

Info Cases: Integrating Use Cases and Domain Models

Michel Heluey Fortuna
(michel.fortuna@ufjf.edu.br)

Cláudia Maria Lima Werner
(werner@cos.ufrj.br)

Marcos Roberto da Silva Borges
(borges@nce.ufrj.br)

Programa de Engenharia de Sistemas e Computação

COPPE / UFRJ

Rio de Janeiro, June 2008

Info Cases: Integrating Use Cases and Domain Models

There is evidence of a gap between use case modeling and domain modeling in the development of a system, particularly during the system requirements definition phase. For example, the level of automation achieved in proposals to generate the domain model from use cases, or to verify the consistency between them, is low or depends on the interpretation of the modeler. Moreover, it has already been seen that different modelers, working independently, produce very different domain models based on use cases of the same system. This report analyzes this problem and proposes a specialization of the use case model to serve as an integrated requirements model from which a domain model can be derived. Semi-formal rules are presented to demonstrate this capacity, as well as results of studies carried out to assess the proposed model.

1 Introduction

In system requirements modeling, it is commonly recommended that the use case model (UCM) and the domain model[1] (DM) be used together. However, there is evidence of a gap between use case modeling and domain modeling in the development of a system. For example, proposals to generate the DM from the UCM [1] [11] [15], or to verify the consistency between them [5] [8], have a low level of automation or else use some form of linguistic analysis, which is known to be incapable of achieving conclusive results. In other words, much depends on the interpretation of the modeler. Further evidence of this gap is the fact that different modelers, working independently, produce very different DMs based on use cases (UCs) of the same system [16]. In particular, difficulties have been observed in identifying concepts of the system domain, in assigning responsibilities to them, and in deciding upon appropriate levels of abstraction.

This report aims to propose a solution to these problems. It begins by pointing out three obstacles to the integration of the models (section 2), then goes on to discuss a way of overcoming them (section 3). Based on this, it presents a proposal for a solution (sections 4 and 5) and studies carried out to assess its effectiveness (section 6). Finally, the report discusses related works (section 7) and gives suggestions for further research (section 8).

[1] The domain layer of software objects [9]

2 Obstacles to the integration of the models

There are three main obstacles to a greater integration of UCM and DM.

Obstacle 1: Use cases (UCs) do not have formal links with the DM.

UCs lack formal elements from which it is possible to obtain, safely and decisively, a single DM, even a preliminary one. The information relevant to the determination of the DM, when it exists, remains “hidden” in the descriptions in natural language of UCs. During the elaboration of UCs, there is no specific concern with the capture and specification of formally identifiable elements that can determine a DM or facilitate the verification of consistency between the models. Consequently, much of the attainment of the DM depends on the interpretation of the modeler. This takes us on to the second obstacle.

Obstacle 2: The modeler’s knowledge about the domain is insufficient.

In general, modelers have relatively little knowledge about the system domain. Their contact with the domain is often limited to a short period of time when they take part in the system specification.

Obstacle 3: The usual criterion for elicitation of UCs is subjective.

The criterion generally recommended for this is that every UC should have value for at least one stakeholder [7] [2] [13]. Unfortunately, this is a very subjective criterion [14], and is partly responsible for the modelers’ difficulty in maintaining suitable levels of abstraction in modeling with UCs.

In principle, the validation of the DM by the stakeholders could be a guarantee against any flaws in it. However, due to the difficulty that, in general, stakeholders have in working with DMs [10], it is very likely that they are induced to “accept” the abstractions that the modeler has put there, in detriment to other more effective ones that could come about if they were to participate more fully in the elicitation and representation of the abstractions.

3 The road to integration

The first two obstacles mentioned in the previous section suggest a course to steer towards integration: the capture, while modeling with use cases and with the direct participation of stakeholders, of formally identifiable elements from which a view of the DM can be derived automatically. That is, the solution must be an integrated model of requirements, rather than the usual approach of a collection of models [5]. Whereas in the collection of models approach each model focuses on one dimension of modeling (behavior, state, structure, etc.) and is treated (created, maintained and used) separately, an integrated model captures several different dimensions in one single conceptual framework from which it is possible to generate views, each view corresponding to one of the modeling dimensions.

Although working with an integrated model may be more complex than using separately each model that it integrates, just the elimination of the problem of consistency between models can represent a very favourable compensation, especially in the system requirements phase, when it is recommended that both models – use cases and domain – be used together to complement each other [5] [8].

To capture the elements for the derivation of the DM, the solution proposed in this report (section 4) uses a device that is little used in use case modeling: the flows of information exchanged between the actors and the system in each UC (*UCs' informational interface*). Since these flows represent the communication between the system and its actors, it is reasonable to expect that the domain abstractions, useful for this communication, are some way represented in these flows. So, if these flows are specified with a minimum of formal rigorousness (e.g. formal syntax, informal semantics), they will reflect these abstractions through their content and structure. Collaborating with the modeler in the specification of these flows, the stakeholders are directly involved in the elicitation of the DM, without going outside the conceptual framework of use case modeling.

Each possible way of partitioning the system in UCs produces a different set of information flows representing the communication between the system and its actors. Therefore, we must choose some partitioning. Two questions immediately arise: 1) what partitioning? and 2) how can we capture and identify, in the flows, the pertinent elements for the construction of the DM?

An answer to the first question can be obtained through a more precise interpretation of the criterion that every UC should have value for at least one stakeholder, normally used to elicit UCs. Such interpretation can be as follows. A UC has value for a stakeholder when it aids the achievement of one of his goals. In turn, achieving a goal means causing a change of state in the system and/or in its environment. The state of the system at the moment when the goal is achieved must be a *steady state*, that is, consistent with the state of the environment in which the system is inserted, and therefore free of any need for rollback to a previous state, even if no other UC be activated subsequently.

It is interesting to note that the requirement that every UC should leave the system in a steady state affects the partitioning of the system into UCs, and that the modeler must judge the steadiness of a state based on his knowledge of the system domain. To demonstrate this, consider a point-of-sale system in a supermarket. In it, the registering and payment of a purchase must form part of the same UC, since if a purchase is registered and not paid for, it will be necessary to return the system to its state previous to registering the purchase (and the products to the supermarket shelves). On the other hand, in a restaurant management system, registering a customer's order and its payment must constitute different UCs, since once the order has been made (and the meal eaten), even if it is not paid for, the record of the order must be kept in the system (because the dishes and the drinks were effectively consumed).

Thus, the partitioning of the system into UCs must be such that the execution of each UC leads to the achievement of some stakeholder's goal and leaves the system in a steady state. This partitioning criterion defines a level of abstraction for the elicitation of UCs which we call *Informational Level of Objectives* (ILO).

An answer to the second question above is given in the next two sections.

4 Info Cases

In this section we introduce the concept of *info case*. An *info case* (IC) is a UC of the ILO, with its informational interface specified in a formalism capable of capturing elements of the DM view of the integrated model, and of permitting the formal identification of these elements. Such a formalism was proposed in [3] [4]. It has two parts:

1) A specification of the composition of the flows, and

2) A dictionary of elementary items of information.

These parts are illustrated in Figure 1 and Table 1 respectively, for a restaurant management system.

In this formalism, the language adopted to describe the composition of the flows of the informational interface is practically the same as that used previously in Structured Analysis [18] [10] to specify data flows and data stores. It employs a small set of symbols for the construction of data expressions.

The signs ➔ and 🡐 indicate, respectively, information *input flows* and *output flows*, from the point of view of the system. A piece of information or *information item* in a flow can be elementary (indivisible) or compound (made up of other elementary or compound items). Compound items are also called *packages* and are indicated by the symbol 🗐. The notation used to describe the composition of a flow or package is the following: **+** means *composition*, $_{n}\{x\}_{m}$ means from *n* to *m* occurrences (or *repetitions*) of x, **( )** is used to group items, **|** means *or* and **[ ]** delimits *conditional items*, that is, which will not always be present (they may not be pertinent, according to the context). *Calculated item* is any elementary item present in an output flow (output item), whose value was calculated by the system (that is, not present in any input flow). An *identifier* (*item*) is a special kind of calculated item. It can be recognized by the name ending with *_id* (for example, *cust_id*), and represent a domain abstraction captured during the requirements modeling.

ACTOR: Customer **IC 1: Make order**

→ order = order_date + table_id + order_items

🗐 order_items = $_1${item_id + item_quant}

← order_id

ACTOR: Customer **IC 2: Cancel order**

→ canc_order = order_id

ACTOR: Customer **IC 3: Request bill**

→ req_bill = order_id + [cust_id]

← bill = order_id + table_no + order_date + bill_items + bill_value + [cust_name + cust_tel]

🗐 bill_items = $_1${item_id + item_type + item_name + unit_price + item_quant + item_value}

ACTOR: Customer **IC 4: Pay bill**

→ payment = order_id + cash_tendered + pay_date

← change

ACTOR: Customer **IC 5: Leave bill pending**

→ pend = order_id + cust_id

ACTOR: Manager **IC 6: Register regular customer**

→ customer = cust_name + cust_tel

← cust_id

ACTOR: Manager **IC 7: Update menu**

→ menu_item = item_name + unit_price + item_type + item_descr

← item_id

ACTOR: Manager **IC 8: Request consumption for day**

→ consump_req = report_date + consump_date

← consump_day = report_date + consump_date + comsumed_items

🗐 consump_items = $_0${item_type + {item_id + item_name + item_quant} }$_2$

ACTOR: Manager **IC 9: Request receipts**

→ rec_req = report_date + rec_period

🗐 rec_period = start_date + end_date

← receipts = report_date + rec_period + consump_value + rec_paid + rec_pending +
rec_servTax + total_rec

ACTOR: Manager **IC 10: Register table**

→ table = table_no

← table_id

ACTOR: Manager **IC 11: Request pending bills**

→ pend_req = report_date + pend_period

🗐 pend_period = start_date + end_date

← pends = report_date + pend_period + {cust_id + cust_name + cust_tel + pend_bills +
custPend_value} + pend_value

🗐 pend_bills = $_1${order_id + order_date + table_no + bill_items + bill_value}

🗐 bill_items = $_1${item_id + item_type + item_name + unit_price + item_quant + item_value}

Figure 1. Informational interface (flows) – Restaurant System

Table 1. Dictionary of elementary items in IC 3: Request bill (partial)

Actor: Customer	**IC 3: Request bill**		
Name	**Description**	**Type**	**Domain**
order_id	Identifier of the order whose bill is requested	Natural no.	
item_quant	Quantity consumed of an item.	Natural no.	
item_value	Value of the consumption of an item. Unit price × quantity consumed of the item.	Currency	
item_type	Type of the consumed item	Text	{dish, drink}

5 Derivation rules for the DM view

This section presents a set of rules for the derivation of the DM view from the info cases (ICs). The aim is to show evidence of the integration between the UCM and the DM, achieved through ICs.

The presentation of the rules is divided into 4 sections - one for each kind of DM element they help to determine: classes, associations, attributes and operations. The specification of the informational interface flows of the Restaurant system (Figure 1) is used to exemplify the application of the rules. The resulting DM is shown in Figure 2.

The main functions of the restaurant system are:

a) To register the order of dishes and drinks by each customer;

b) To print the bill when the customer asks for it;

c) To cancel an order at the request of the customer;

d) To register the payment of a bill or the fact that it is left pending (leaving the bill pending is only allowed for customers that are considered regulars);

e) To print a detailed report of the daily consumption of dishes and drinks, in order to replace stocks;

f) To print a detailed report of the restaurant's receipts in a given period, separating receipts effectively paid from those payable (bills left pending); and

g) To register the menu that is in effect, allowing it to be printed at the request of the manager.

5.1 Classes

Classes are determined by the identifiers generated in the ICs. A *generated identifier* in an IC is an identifier whose value is established during the execution of the IC. Its purpose is to serve as a reference to an object. For example, *cust_id* is a generated identifier in IC 6 (*Register regular customer* - Figure 1) to serve as a reference to an object (*customer*) created during the processing of the IC.

Rule 1 (classes). Each generated identifier produces a class.

The classes obtained from the generated identifiers represent abstractions used and shared by the domain specialists who take part in the definition of the system requirements. Thus, such classes are likely to be useful, with significant attributes and operations in the domain [12].

In the Restaurant system (Figure 1): *order_id*, generated in IC 1 - *Make order*, determines the class *Order*. The other classes determined in this way are: *Customer* (*cust_id*, IC 6), *Item* (*item_id*, IC 7) and *Table* (*table_id*, IC 10).

5.2 Associations

The relationships among the many objects manipulated by the system are represented in the informational interface of the ICs, since the system actors must necessarily inform them. For example, in the Restaurant system, the relationship between an order and its table must be informed to the system.

Since the objects are represented by their identifiers, the relationships among objects are represented by the occurrence of two or more identifiers in the informational interface of each IC.

Rule 2 (associations). Associations are indicated by the occurrence of more than one *identifier* in the informational interface of an IC[2] (for output flows only generated identifiers should be considered)[3].

In principle, each possible pair of identifiers may indicate an association to include in the DM. In the Restaurant system, for example, the pairs <*order_id*, *table_id*> and <*order_id*, *item_id*>, obtained from IC 1, and the pair <*order_id*, *customer_id*> from IC 3 or IC 5, determine the associations *Order-Table*, *Order-Item* and *Order-Customer*, respectively.

[2] Since all references to objects are made by means of identifiers.

[3] So as to obtain only new associations.

5.3 Attributes

The activation of ICs in the Informational Level of Objectives (ILO) depends on the initiative of some actor, and therefore, no general assumption can be embedded into the system regarding the coexistence, in time, of the processes underlying the ICs. Consequently, every piece of information which needs to be communicated among these processes must be considered state information, that is, information to be persisted between system states attained by the execution of the processes. Besides, as these states are always stable[4], those pieces of state information will become attributes of system domain classes. This results in the following rule.

Rule 3a (attribute determination). Every non-identifier elementary item, input in an IC and needed in another IC must be persisted; otherwise, it should not be persisted. The same holds for every non-identifier calculated item in the output flow of an IC, whose originally calculated value is needed in another IC and cannot be restored in that IC.

In the Restaurant system (Figure 1) there are many input items. For example, the elementary item *order_date* enters in IC 1 (input flow *order*) and has its value retrieved in ICs 3 and 11. Therefore, it must be persisted. On the other hand, *cash_tendered* and *item_descr* do not have to be persisted; the former because it is used only in the same IC in which it entered (IC 4, to calculate the change), and the latter because it is not used in any IC, not even in the one in which it entered the system (IC 7) (probably a mistake that this rule helps to uncover). For the output items, consider, for example, the output (non-identifier) item *item_value* calculated in IC 3. It has to be retrieved in IC 11 in order to print out the pending bills. But it can not be guaranteed that the value of *item_value*, calculated at the moment when the customer asks for the bill (IC 3), can be restored later when the manager requests the list of pending bills (IC 11). This is because, in the meanwhile, the unit price of the item may be changed (through IC 7 - *Update menu*). Therefore, the *item_value* calculated in IC 3 must be persisted.

Once the persistence of an item is decided upon, the next step is to allocate the item as an attribute of one of the classes determined with rule 1 (section 5.1). Modeling with ICs, the only way to establish a link between an item and the object of which it represents a property (attribute) is to include the identification of the object in the same informational interface that contains the item. That is what happens, for example, in the informational interface of IC 4, which, besides *pay_date*, includes *order_id* to identify the order that is being paid. The following rule establishes this. It uses the concept of *classes attained by*

[4] Consistent states that can persist indefinitely.

an identifier, which includes: a) the class it identifies, and b) the association classes (if there are any) in which it takes part as one of the identifiers.

Rule R3b (attribute allocation). Each item to be persisted, present in the informational interface of an IC, must be allocated as an attribute of one of the classes attained by the identifiers present in the same interface.

In the Restaurant system, for example: *order_date* and *item_quant* (IC 1) must be allocated, respectively, to classes *Order* (identified by *order_id*) and *Order-Item* (association class attained by *order_id*); *item_value* (IC 3) must be allocated to the class *Order-Item* (association class attained by the identifier *order_id*); and *pay_date* (IC 4) to the class *Order*; etc.

The modification of the system state, i.e. of an object or of an association among objects, commonly occurs in ICs whose informational interface contains some information item to be persisted. However, even ICs without this characteristic may produce a state change. That is because the mere occurrence of the IC's dispatching event may imply the change. Obviously, this change is also implemented by means of an attribute of the object (or of the association), which can be an attribute previously created to persist an item of the interface, or an attribute specially created to reflect the change. The latter will be distinguished from other (ordinary) attributes by the term *state attribute*. Thus, unlike ordinary attributes that directly correspond to persisted items present in the input and output flows of ICs, state attributes are not shown in the informational interface of ICs. Despite this, in some cases it is still possible to derive from that interface the need of this special kind of attribute (rule R3c).

Rule R3c (state attributes). In ICs with an informational interface made up of only an input flow containing just identifiers, a state attribute must be added in one of the classes attained by those identifiers.

In the DM of the Restaurant system (Figure 2), the attributes *cancelled?* and *pending?* were introduced to the class *Order*, through the application of this rule to ICs 2 (*Cancel order*) and 5 (*Leave bill pending*), respectively.

5.4 Operations

The operations to be included in the DM must allow:

a) The accomplishment of state changes in the system by the creation and change of objects (rules 4a and 4d); and

b) The accomplishment of state changes in the system environment, by producing the outputs foreseen in the informational interface, represented by the non-persisted calculated items and output flows (rules 4b and 4c).

Rule R4a (constructors). Every generated identifier produces a constructor operation for the class it identifies.

In the Restaurant system, this rule gives rise to 4 constructor operations: *Order()* (IC 1), *Customer()* (IC 6), *Item()* (IC 7) and *Table()* (IC 10), in the respective classes of the same names.

Rule R4b (calculated items). Every non-persisted calculated item produces an operation to calculate it[5].

In the Restaurant system, this rule produces the following operations: *bill_value()* from IC 3, *change()* from IC 4, *item_quant()* from IC 8, *consump_value()*, *rec_paid()*, *rec_pending()*, *rec_servTax()* and *total_rec()* from IC 9, *pend_value()* and *custPend_value()* from IC 11.

Rule R4c (flows). Every output flow present in an IC whose processing does not produce a system state change (typically *reporting ICs*) and which is not made up of just one non-persisted item[6], causes an operation to produce the flow.

In the Restaurant system, *consump_day()* (IC 8), *receipts()* (IC 9) and *pends()* (IC 11) are the resulting operations.

Rule R4d (state change). Every IC that causes system state change yields an operation to perform it and to produce the output flow (if any), unless a constructor operation, resulting from rule R4a, performs all the change by itself and there is no output to produce.

In the Restaurant system, rule R4d gives rise to four operations: *cancel()* (IC 2), *requestBill()* (IC 3)*, payBill()* (IC 4), and *leaveBillPending()* (IC 5).

Rule R4e (allocation to classes). The operations produced by rules R4b, R4c and R4d are to be included in one of the classes attained by the identifiers present in the informational interface of the IC, or if there are none, to be included in the class that represents the system (to keep the cohesion of the remaining classes).

[5] Persisted calculated items do not generate operations, since they are available in the class interface as attributes.
[6] Already dealt with in rule R4b.

The class that represents the system can be seen as a repository of operations to be transferred to control classes introduced in a later phase of the system analysis. All other classes of the DM tend to have high semantic cohesion, since they correspond to abstractions used by the stakeholders in their daily business. Similarly, the coupling among classes tends to reflect the relationships that exist among the domain abstractions.

The visibility of the operations must be public to the actors that are allowed to enact the corresponding ICs, since the operations come directly from the system requirements model. The operations determined by the non-persisted calculated items (rule R4b) return the value calculated for the item. Most of the operations' arguments can be obtained from the items present in the input flow of the ICs. The specification (type and initial value) of the arguments is based on type and domain information of the corresponding item, which appears in the dictionary of elementary items. For example, from the informational interface (Figure 1) and the dictionary of IC 3 (Table 1), we obtain the specifications: *requestBill(in cust: Customer)* and *bill_value(): Currency*. Figure 2 shows the resulting DM for the *Restaurant* system.

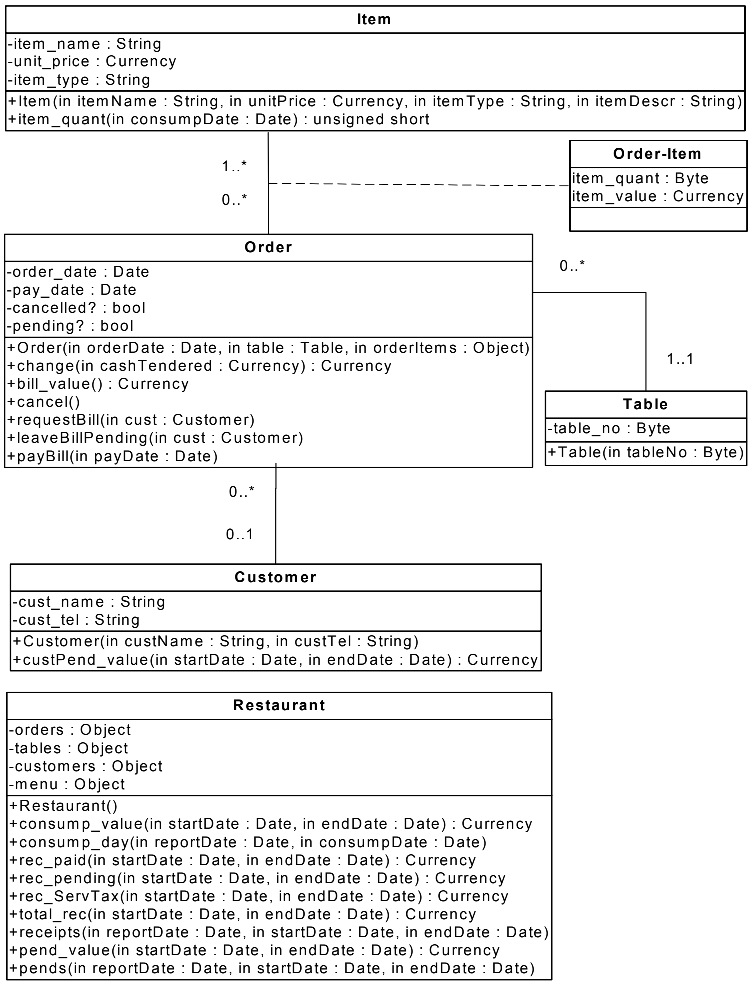
Item
-item_name : String
-unit_price : Currency
-item_type : String
+Item(in itemName : String, in unitPrice : Currency, in itemType : String, in itemDescr : String)
+item_quant(in consumpDate : Date) : unsigned short
1..*
0..*
Order-Item
item_quant : Byte
item_value : Currency
Order
-order_date : Date
-pay_date : Date
-cancelled? : bool
-pending? : bool
+Order(in orderDate : Date, in table : Table, in orderItems : Object)
+change(in cashTendered : Currency) : Currency
+bill_value() : Currency
+cancel()
+requestBill(in cust : Customer)
+leaveBillPending(in cust : Customer)
+payBill(in payDate : Date)
0..*
1..1
Table
-table_no : Byte
+Table(in tableNo : Byte)
0..*
0..1
Customer
-cust_name : String
-cust_tel : String
+Customer(in custName : String, in custTel : String)
+custPend_value(in startDate : Date, in endDate : Date) : Currency
Restaurant
-orders : Object
-tables : Object
-customers : Object
-menu : Object
+Restaurant()
+consump_value(in startDate : Date, in endDate : Date) : Currency
+consump_day(in reportDate : Date, in consumpDate : Date)
+rec_paid(in startDate : Date, in endDate : Date) : Currency
+rec_pending(in startDate : Date, in endDate : Date) : Currency
+rec_ServTax(in startDate : Date, in endDate : Date) : Currency
+total_rec(in startDate : Date, in endDate : Date) : Currency
+receipts(in reportDate : Date, in startDate : Date, in endDate : Date)
+pend_value(in startDate : Date, in endDate : Date) : Currency
+pends(in reportDate : Date, in startDate : Date, in endDate : Date)

Figure 2. DM view for the Restaurant system

6 Experimental studies with info cases

Several experimental studies have already been carried out to assess modeling with ICs [3] [4]. Three of these, focussing more specifically on the derivation of the DM, are reported in the following.

6.1 Study 1: Proof of concept

This case study involved a team with 1 requirements engineer, 4 programmer-analysts and 4 domain specialists in the development of a real multi-user web system , to support the activities of a company in the field of medicine and safety at work. The system was developed by the analysts in PHP and MySQL from the IC model elaborated by the engineer with the support and validation of the specialists. It was made up of 30 ICs, with a total of 40 pages of specifications. The development took four months of work.

The system DM was elaborated by the analysts together, on an *ad hoc* basis, since the study preceded the definition of the rules presented in section 5. Even so, the analysts elaborated the system DM with no difficulties. At the end of the study, all those involved had a positive impression regarding ICs and the resulting system. Over the 3 years that it has been in continuous operation, the system has had normal maintenance, most of it for evolution.

Some aspects that show the importance of the results obtained are:

a) None of the analysts had previous knowledge of, or experience in IC, nor in the system domain;

b) The analysts worked for the first three months practically without personal contact with the requirements engineer, communication between them being carried out in writing via instant messenger-type software;

c) The specifications were elaborated in parallel with the development of the system, each IC being released as soon as its definition was ready;

d) The division of work among the analysts was all done by IC;

e) The DM was elaborated incrementally (at each new IC) by the analysts, without the participation of the engineer;

f) The DM elaborated by the analysts almost always met the expectations of the engineer; occasional discrepancies were easily credited to initiatives inconsistent with the IC model.

6.2 Study 2: Adjustments to the rules and preliminary test of uniformity of DMs

The second study was informal, carried out to preliminarily assess the comprehensibility and usability of the rules of derivation (section 5), as well as to provide a first assessment of the degree of uniformity in DMs resulting from them. Two analysts, graduates in Computer Science, both with two years experience in systems development and one year in reading ICs, took part in the study at different times.

First of all, one of the analysts applied the rules using the specification of a library system with 19 ICs. The DM obtained was compared with the DM previously constructed by the experimenter using the same rules. The discrepancies detected revealed the analyst's difficulties in understanding some rules, which brought about a revision of the rules manual. Later, the manual having been revised, the other analyst did the same exercise. To assess the uniformity of the results, the discrepancies between the model obtained by each analyst and that constructed by the experimenter were counted. For the purposes of this assessment, extra or missing elements (classes, attributes, associations and operations) with regard to the model produced by the experimenter were considered to be discrepancies. Table 2 shows the results of this assessment.

Table 2. Results of Study 2

		Analyst 1		**Analyst 2**	
Elements	**#E**	**#D**	**%S**	**#D**	**%S**
Classes	5	0	100	0	100
Associations	11	1	91	0	100
Attributes	20	4	80	0	100
Operations	41	4	90	1	97

Key	
#E:	No. of elements in the experimenter's model.
#D:	No. of discrepancies.
%S:	% of success in reproducing the experimenter's model

6.3 Study 3: Second test of uniformity of DMs

The aim of this study was to compare the two techniques - ICs and UCs, regarding granularity and uniformity of the DMs produced based on each of them.

The *granularity* of a DM is given by the number of classes that the model has. *Uniformity* measures the degree of similarity between models. In this study, two types of uniformity between models were considered: 1) *conceptual uniformity*, based on the comparison of the concepts (abstractions) existing in each model, at the purely semantic level; and 2) *representational uniformity*, based on the comparison of attributes, associations and operations used to represent each abstraction present in the models.

The uniformity of a type of element (abstractions, associations, attributes and operations) between two DMs was calculated as the ratio between the number of matching occurrences of the element in both models and the number of unique occurrences of the element in the models. For example, the uniformity of abstraction between two different models i and j is given by the formula:

$$\boxed{\text{Unif-A}_{i,j} = \frac{A'_{i,j}}{(A_i + A_j) - (A'_{i,j})}}$$

where, for $i \neq j$:

- A_i is the cardinality of the set of abstractions of Model i;
- A_j is the cardinality of the set of abstractions of Model j; and
- $A'_{i,j}$ is the cardinality of the set of abstractions common to both models i and j.

To determine the uniformity of associations, attributes and operations, only the classes that represent abstractions common to both models i and j were considered.

The representational uniformity between two models i and j was calculated as the arithmetic mean of the uniformities of attributes, operations and associations, between the models:

$$\boxed{\text{Unif-R}_{i,j} = \frac{\text{Unif-At}_{i,j} + \text{Unif-O}_{i,j} + \text{Unif-L}_{i,j}}{3}}$$

The hypotheses tested in the study are:

- H-G: the granularity of DMs built up from ICs is less than the granularity of DMs built up from UCs;
- H-U_a: The uniformity of abstractions between DMs built up from ICs is greater than the uniformity of abstractions between DMs built up from UCs;
- H-U_l , H-U_{at} , and H-U_o: Similar to H-U_a , for associations, attributes and operations, respectively.
- H-U_r: The representational uniformity of DMs built up from ICs is greater than the representational uniformity of DMs built up from UCs.

The participants of the study, 6 professionals – all Computer Science graduates – were divided into two groups of 3. Each group used only one of the techniques – IC or UC – to elaborate the behavioral model and the respective DM. The participants in group A (ICs) were designated by the numbers 1, 2, and 3 – those in group B (UCs) were designated by the numbers 4, 5, and 6. Participant 1 (P1) already had some experience with ICs. All of them had knowledge of and some experience with UCs.

Some days before the study, the participants received training material in the respective technique. On the day of the study they attended a training session and were given a summary of the main functions of the system to be modeled – a financial control system. During the modeling, which took approximately one and a half days, they had free access to the experimenters (the authors of the summary), who played the role of domain specialists. Only participant 3 (P3) had any previous knowledge in this area.

Both groups worked at the same time, in separate rooms, under the supervision of the experimenters, so as to, among other things, avoid any exchange of information among the participants, which could be prejudicial to the results. When the modeling was finished, the experimenters interviewed the participants in pairs (P1-P2, P1-P3, P2-P3, P4-P5, P5-P6, and P4-P6), mainly in order to identify the common abstractions in their DMs, and based on them, identify the attributes, operations and associations used in the representation of these common abstractions.

Tables 3, 4, and 5 show the numbers found for the study variables, for each group – A (ICs) and B (UCs) respectively.

Table 3: Count of classes - Groups A and B

	Group A			Group B		
Partic. / Elem.	P1	P2	P3	P4	P4	P6
Classes	12	7	9	13	9	8

Table 4: Count of the study variables - Group A (ICs)

Pairs / Elem.	P1-P2			P1-P3			P2-P3		
	P1	P2	M[7]	P1	P3	M	P2	P3	M
Abstractions	12	9	8	12	11	10	7	9	6
Associations	5	6	3	10	7	6	6	4	4
Attributes	28	24	15	38	24	16	24	14	10
Operations	7	7	3	7	7	6	10	8	6

[7] Matching occurrences

Table 5: Count of the study variables - Group B (UCs)

Pairs / Elem.	P4-P5			P4-P6			P5-P6		
	P4	P5	M	P4	P6	M	P5	P6	M
Abstractions	13	9	7	13	8	6	9	8	5
Associations	4	5	3	2	7	2	2	4	1
Attributes	34	22	14	28	33	18	24	14	9
Operations	24	16	7	17	15	1	14	12	2

To test the hypothesis of granularity (H-G), the mean of the granularity of the DMs of each group was calculated, which gave the values 9.3333 and 10, respectively, for groups A and B. Therefore the hypothesis was confirmed: DMs produced from ICs have 8% lower granularity than DMs produced from UCs.

The hypotheses of uniformity referring to each element of the DM (abstractions, associations, attributes, and operations) were tested based on the mean of each group for the uniformities of that element, obtained from pairs of models of the group. Similarly, the hypothesis of representational uniformity (H-U_r) was tested based on the mean of each group for the representational uniformities, obtained from pairs of models of the group. Table 6 shows and compares the uniformity means obtained in each group (technique).

Table 6: Comparison of uniformity - ICs vs. UCs

Techique / Elem.	ICs	UCs	Comparison
Abstractions	0,6615	0,4278	ICs + 55%
Attributes	0,3701	0,3541	ICs + 5%
Associations	0,5291	0,3286	ICs + 61%
Operations	0,5076	0,1092	ICs + 365%
Representational	0,4689	0,2640	ICs + 78%

Therefore, Table 6 shows that the DMs produced from ICs were more uniform with each other than those based on UCs. This was seen on the semantic (conceptual) level by the comparison of the abstractions present in the models, as well as at the representational level by the comparison of the associations, attributes and operations present in the classes corresponding to the abstractions that coincide in the models.

7 Related works

There are many proposals to build a DM from the UCM (e.g. [1] [11] [15]), or to establish the consistency between the two models (e.g. [5] [8]). However, these proposals do not follow the approach of an integrated requirements model, but rather they keep the two models – UCM and DM, separate. As mentioned previously (section 1), the result is, invariably, incomplete DMs or a great deal of dependence on the modeler to interpret the elements of each model.

Like us, Glinz [6] also adopts the approach of an integrated requirements model, but his proposal differs from ours in various aspects. It does not use the UCM as the basis for integration. Besides, it attempts a wider integration, involving structure, data, behavior, interaction with the user, etc., aiming at the generation of diverse views, in different levels of abstraction.

Svetinovic [17] tries to solve the problem of inconsistency among DMs obtained by different modelers for the same system. He proposes an alteration of the traditional process of the Object Oriented Analysis ("that observes only concepts"), to adopt an approach focused on activities, which attempts to identify the concept responsible for a particular activity. This way, he aims to validate the concepts through an "activity-purpose" analysis, and to obtain more restricted intermediary artifacts than the DM, capable of providing more consistency among the DMs. Therefore his strategy is to proceed with the analysis before attempting a conceptual decomposition, in order to restrict the freedom in choosing the domain abstractions and its responsibilities. As we see it, one of the features of this strategy is to keep the main responsibility to identify the concepts with the modeler, that is, let him decide, based on various artifacts (and not only on UCs), what concepts to choose to compose the DM. Differently, our strategy assumes that the modeler should first of all consider the concepts that the stakeholders use in their every-day business. Ideally, these concepts should be captured during the requirements elicitation. In this way, the UCM serves as a channel to elicit the domain concepts with the active participation of the stakeholders. The modeler takes this opportunity to exploit the familiarity of the stakeholders with the UCM to extract from them the concepts that will constitute a kind of baseline for the DM. It is also possible for the modeler to suggest new abstractions or improvements to the abstractions used by the stakeholders, working from the basis of the abstractions consolidated by them.

8 Conclusion and Future Works

This report pointed out three obstacles to a greater integration of the UCM with the DM: 1) a lack of formal links between UCM and DM; 2) insufficient knowledge of the domain by the modelers; and 3) subjectivity of the usual criterion for the elicitation of UCs. These obstacles make it difficult to obtain suitable solutions to increase the degree of automation (or reduce the dependence on the modeler) in the generation of the DM from the UCM, or in the verification of the consistency between them. Moreover, in our view, these are the main reasons for the lack of consistency between DMs obtained by different modelers for the same system.

To overcome these obstacles, the report proposes an integrated requirements model built up from info cases (ICs) – a specialization of UCs resulting from the fixing of a special level of abstraction for their elicitation and greater detailing and formal meaning in the specification of the information flows exchanged between the system and its actors.

As evidence of the degree of integration achieved and of the capacity of ICs to overcome the mentioned obstacles, semiformal rules were presented for deriving a DM from the UCM. In addition, three experimental studies were reported, which showed evidence of the usability of the rules and their capacity to solve (or to reduce) the problem of inconsistency among DMs.

Future works that are planned or suggested include:

1) Further experimental studies, this time with a larger number of participants (probably students) in order to confirm the hypotheses of uniformity among DMs with statistically significant results, and to make a comparative assessment of ICs and UCs as to the effort required in modeling, training needs and the quality of the models obtained;
2) Studying the possibility of automatically propagating alterations made in the DM to the UCM;
3) Studying the contribution of ICs in the context of agile software development methods and in the context of model-driven development;
4) Characterizing with precision the level of automation obtained with the rules; and
5) Developing a tool to support the application of the rules.

Acknowledgement. We thank Leonardo G. P. Murta, Tarcísio S. Lima and Carlos Henrique C. Duarte for comments on the subject of this report. Special thanks are due to Vitor Padilha Gonçalves for his active participation in the planning and development of Study 3. The first author was partially supported by a scholarship from CAPES, Brazil.

9 References

[1] Biddle, R., Noble, J., and Tempero, E., "From Essential Use Cases to Objects"*, 1st Intl. Conf. on Usage-Centered, Task-Centered, and Performance-Centered Design* (forUse 2002), Ampersand Press, Rowley, MA, 2002, pp. 1-23.

[2] Bittner, K., and Spence, I., *Use Case Modeling*. Addison-Wesley Canada, 2002.

[3] Fortuna, M. H., *Um Modelo Integrado de Requisitos com Casos de Uso*, PhD qualification report, COPPE/UFRJ, Rio de Janeiro, October 2006 (In Portuguese).

[4] Fortuna, M. H., Werner, C. M. L., and Borges, M. R. S., "Um Modelo Integrado de Requisitos com Casos de Uso", *X Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software* (IDEAS'07), Isla de Margarita, Venezuela, May 7-11, 2007, pp. 313-326 (In Portuguese).

[5] Glinz, M., "A Lightweight Approach to Consistency of Scenarios and Class Models", *4th Intl. Conf. on Requirements Engineering* (ICRE'00), Illinois (USA), June 2000, pp. 49-58.

[6] Glinz, M., Berner, S., Joos, S. "Object-Oriented Modeling with ADORA", *Information Systems* 27, 6, 2002, pp. 425-444.

[7] Jacobson, I., "Use cases - Yesterday, Today, and Tomorrow", *Software and Systems Modeling 3*, *3*, Springer Berlin, 2004, pp. 210-220.

[8] Kösters, G., Six, H-W., and Winter, W., "Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications". *Requirements Engineering Journal 6*, Springer London, 2001. pp. 3-17.

[9] Larman. C. *Applying UML and Patterns* (3rd ed.), Prentice Hall, New Jersey, 2004.

[10] Lauesen, S.: *Software Requirements - Styles and Techniques*. Addison-Wesley London, 2002.

[11] Liang, Y.: “From se Cases to Classes: A Way of Building Object Model with UML”. *Information and Software Technology 45*, Elsevier, 2003. pp. 83-93.

[12] Meyer, B.: *Object-Oriented Software Construction* (2nd ed.), Prentice Hall PTR, New Jersey, 1997.

[13] Object Management Group (OMG), *UML Superstructure Specification, v2.0*. OMG document formal/05-07-04. (Available at http://www.omg.org/cgi-bin/doc?formal/05-07-04)

[14] Robertson, S., and Robertson, J., *Mastering the Requirements Process*, 2 ed., Addison Wesley Professional, New Jersey, 2006.

[15] Subramaniam, K., Liu, D., Far, B. H., and Eberlein, A., “UCDA. Use Case Driven Development Assistant Tool for Class Model Generation”, *16th Intl. Conf. on Soft. Eng. & Knowledge Eng.* (SEKE’04), Canada, June 2004, pp. 324-329.

[16] Svetinovic, D., Berry, D., Godfrey, M., “Concept Identification in Object-Oriented Domain Analysis: Why Some Students Just Don’t Get It”, *13th IEEE Intl. Conf. on Requirements Engineering* (RE’05), Paris, France, September 2005, pp. 189-198.

[17] Svetinovic, D., “*Increasing the Semantic Similarity of Object-Oriented Domain Models by Performing Behavioral Analysis First*”. PhD. Thesis, University of Waterloo, Ontario, Canada, September 2006.

[18] Yourdon, E. *Modern Structured Analysis,* Prentice Hall PTR, USA, 1988.