Improving Use Case Based Requirements Using Formally Grounded Specifications

C. Choppy[1] and G. Reggio[2]

[1] LIPN, Institut Galilée - Université Paris XIII, France
[2] DISI, Università di Genova, Italy

Abstract. Our approach aims at helping to produce adequate requirements, both clear and precise enough so as to provide a sound basis to the overall development. We present a technique for improving use case based requirements, by producing a companion Formally Grounded specification, that results both in an improved *requirements capture*, and in a *requirement validation*. The Formally Grounded requirements specification is written in a "visual" notation, using both diagrams and text, with a formal counterpart (written in the CASL-LTL language). The resulting use case based requirements are of high quality, more systematic, more precise, and its corresponding Formally Grounded specification is available. We illustrate our approach on an Auction System case study.

1 Introduction

While tools and techniques are now available to support quite efficiently software development, one of the most difficult part remains to produce adequate requirements, both clear and precise enough so as to provide a sound basis to the overall development.

Formally based specifications are advocated since they lead to precise, unambiguous descriptions, but they remain difficult to use and impractical in quite a number of cases. We think the reason for this is twofold. One point that was often put forward is the difficulty to write and read such specifications. Another point we see is that it may be difficult to start with formal specifications while still working on the requirements (thus, trying to understand what is the problem about), hence our idea to take advantage of use cases.

Use cases were introduced by Jacobson [10] after the earlier idea of scenarios, which are the different possible courses that different instances of the same use case can take. Use cases are used to describe/capture the requirements of software systems, while providing an overall picture of what is happening in the system. The use case description is textual (it should be "familiar", easy to read) and sums up a set of scenarios.

Use cases are popular because they are easy to use and informal, however "use cases are wonderful but confusing" [7]. A both good and bad thing is that there is a lot of freedom in what should include a use case description, and how it should be written. UML [17] proposes a diagram for use cases, states that descriptions are needed too, and that the sequence of use case activities are documented by behaviour specifications (e.g., with interaction diagrams). However, examples show that use cases are often imprecise, and also that the terms used are vague or ambiguous.

Since use cases are written in the early phases of software development, it is crucial that they should be worked out with a lot of care, so as to avoid to generate errors that will be difficult and costly to correct further on. Interesting work is done to propose some guidelines on how to write use case descriptions, e.g., Cockburn[7] proposes *templates* for structuring their descriptions. In the following, we use an adaptation of this template provided by Sendall[13].

Our idea is to find a way to combine both advantages of use cases and of formal specifications. Here, we present a technique for improving use case based requirements, developing a companion Formally Grounded specification, that results both in an improved *requirements capture* (some requirements may be updated and some may be new), and in a *requirement validation* since writing the specification leads to check that the requirements can be further made explicit up to a precise specification.

The produced requirements specification is written in a "visual" notation, using both diagrams and text, with a formal counterpart which is written in the CASL[11] and CASL-LTL[12] specification languages.

Being Formally Grounded, our method is systematic, and it yields further questions on the system that will be reflected in the improved use case descriptions. The resulting use case descriptions are of high quality, more systematic, more precise, and their corresponding Formally Grounded specification is available.

In Sect. 2 we shortly sum up our Formally Grounded approach for writing the requirements specification of a software system (see [6] for a full presentation with other examples). In Sect. 3 we present our method for improving use case based requirements using Formally Grounded specifications, and in Sect. 4, we then show how our method applies to a part of the Auction System case study (the complete version is in [5]), showing how the starting use case based requirements have clarified, and how many relevant aspects of the Auction System have been enlightened, before concluding and discussing some related work in Sect. 5.

2 Our Formally Grounded Approach for Requirement Specification

Our Formally Grounded specification approach (see [6] for a complete presentation), aims at helping the user to understand the system to be developed, and to write the corresponding formal specifications. We also support visual presentations of formal specifications, so as to "make the best of both formal and informal worlds". We developed this method for the (logical-algebraic) specification language CASL [11] (Common Algebraic Specification Language, developed within the joint initiative CoFI[1]), and for an extension for dynamic systems CASL-LTL[2][12]. Hence, for each visual specification, its formal counterpart in CASL or CASL-LTL is given.

Our method caters for three different kinds of modelling/specification entities, (i) a data structure, or data type, (ii) a simple dynamic system, that is a single dynamic entity, and (iii) a structured dynamic system, that is composed of mutually interacting dynamic entities; while keeping a common "meta"-structure and way of thinking.

Each entity considered may be modularly decomposed - so its (sub)*parts* are identified-, and is characterized by its *constituent features*. Its model/specification consists of a visual presentation of these parts and constituent features, and of their properties

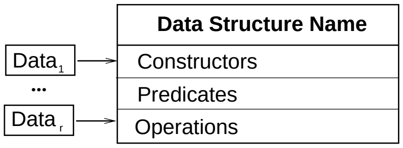
[1] http://www.cofi.info

[2] LTL stands for Labelled Transition Logic[8, 2].

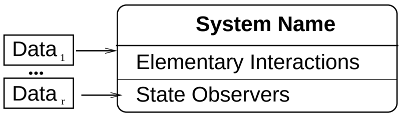
expressed in a natural-language style notation based on an appropriate underlying logic (the variant of logic depends on the kind of entity).

Once the constituent features are identified, we provide guidelines for an exhaustive search of the properties. To this end, we use a tableau whose cells, indexed by the pairs of constituent features, should be filled. For each cell we give a schemata for the relevant properties it should contain, expressing either, when the two indexes are different, the mutual relationships between the two features, or, when they are equal, what is known on that feature. This tableau-filling method ensures that no crucial part of the specification is forgotten, and results in producing a quite structured/navigable set of properties, which should be suitable to support evolution.

Data Structures Data structures are characterized by a set of values, some constructors to denote those values, and some predicates and operations. Data structures may be structured, e.g., they may import other data structures. These features and the imported data structures (the parts) are visually presented as in the diagram below. Their properties are expressed in a many-sorted, first order logic [11] with a natural language-like notation. The tableau-filling technique provides a systematic way to find the respective properties of constructors, predicates and operations, e.g., definedness, truth of predicate, etc., see [6].



Data Structure



Simple Dynamic System

Simple Dynamic Systems Simple dynamic systems are characterized by their states and their transitions, where each transition corresponds to a change of state together with a set of elementary interactions with the external world. Each elementary interaction is described by a name and possibly by parameters (data values). The states are abstractly characterized by "state observers", which, given some parameters, may return some value (operations) or the truth of some condition (predicates). Thus, the constituent features of a simple (dynamic) system are its *elementary interactions* with the external world and its *state observers*. The parts of a simple system are its *data structures* needed to define the parameters and the results of elementary interactions and state observers. The above diagram visually presents which are the subparts and the constituent features, while the specification of the parts is given separately. These properties are expressed with a natural language -like notation derived by Casl-Ltl. Casl-Ltl [12] is a Casl extension based on LTL (Labelled Transition Logic) [8, 2], a branching time temporal (many-sorted, first-order with edge formulae) logic. This notation uses combinators for expressing that elementary interactions take place (e.g., e **happened**), standard logics (**if** - **then** -**else**, **not**, **and**, $=$, **exists**, ...), and also temporal combinators (**next**, **eventually**, **before**, **in any case**, **in one case**[3]).

A transition from one state to another is characterized by elementary interactions, and properties about states and transitions are expressed, e.g., pre- and post-conditions for elementary interactions, or incompatibilities between them. Properties for a state

[3] The last two express universal and existential quantification over execution paths.

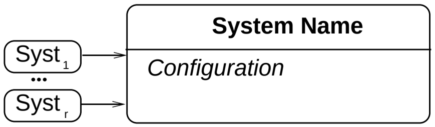
observer explore e.g., which elementary interactions may cause a change in its value, or which are its possible changes. Again, the tableau-filling technique provides a systematic way to find these properties.

Both diagrams and text have a formal counterpart in the CASL-LTL language [12].

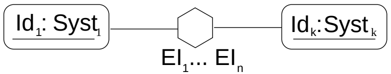
Structured Dynamic Systems A structured (dynamic) system is a specialized simple system that is composed of several dynamic systems, its *subsystems*, which can in turn be simple or structured. A transition of such a system should reflect which are the subsystems transitions that occur. Moreover, it is necessary to describe how the subsystems synchronize.

We present here a simpler version of structured systems (the general case is given in [6]) that have only simple subsystems (possibly of different types), where two subsystems may interact only pairwise by performing simultaneously the same elementary interaction, i.e., the behaviour of these structured systems is given by transitions made of groups of subsystem transitions, where each elementary interaction of a subsystem is matched by one of another subsystem. Furthermore, the considered structured systems are closed, i.e., they have no interaction with the external world.

A structured system is visually presented by :
(i) a Context View which is a *configuration diagram* showing the *subsystems* (in the *Configuration*) and their types (the specifications of all those types are given separately), accompanied by a *cooperation diagram* showing the *cooperations* among the subsystems (each cooperation is given by the synchronized execution of elementary interactions, say EI_i).



Configuration Diagram



Cooperation Diagram

(iii) a Data View which puts together the specifications of all *data structures* that are parts of the system and of its subsystems.

To specify the requirements on a software System it is sufficient to specify a structured dynamic system, whose subsystems are the System itself and all those entities interacting with it (context entities). The specification of the System will be the requirements, whereas the specifications of the context entities will show the assumptions made by the System on the context entities.

3 The Method for High-Quality Requirements

We present in this section our method for producing enhanced requirements. It is organized in five tasks, and works on use case based requirements while developing a companion Formally Grounded specification, which results in improved requirements.

Task 1 Give the use case based requirements on the System following the method proposed by S. Sendall and A. Strohmeier in [14].

Task 2 Find out which are the external entities playing the roles corresponding to the various actors (*context entities*) and determine their types. At this point we can draw a first version of the Context View (see end of previous section), by depicting the System and the found context entities together with their types; the cooperation diagram will have only arcs connecting the System with the context entities.

Task 3 By examining the use case descriptions, one after the other, look for *elementary interactions* and *state observers* of the System; the former should model the interactions between the System and the actors appearing in the use case scenarios, whereas the latter should model information recorded in the System examined or updated in the use case scenarios. Both of them should be depicted in the visual presentation of the System specification (together with the type of their arguments and/or results); the elementary interactions should also be reported in the *cooperation diagram* to show which context entities are taking part in that interactions.

In the meantime put in the Data View any data structure that is used as an argument or a result by a found elementary interaction or state observer. The association between use cases and the elementary interactions and state observers related to it (i.e., which are needed to describe it) should be recorded. During this task, it is possible to find *new entities* interacting with the System that do not correspond to already known actors; they should be added to the Context View, together with their specifications. Whenever, there are relevant assumptions on the context entities they should be made explicit by giving their Formally Grounded specification (they are just simple dynamic systems).

Task 4 Find all the properties about the System following our tableau-filling method. When filling a cell related to some constituent features (elementary interactions and state observers), the descriptions of the associated use cases should be examined as a source of inspiration. During this task, probably, new state observers and data structures will be added, and perhaps the parameters of the existing elementary interactions and state observers may be modified.

Task 5 During the tasks 3 and 4 many *questions* about the System will arise, many aspects of the System that need to be investigated will be highlighted, and many aspects of the System precisely described by the use cases will be found not convincing. These points may be settled following the usual ways, e.g., by interacting with the client, if available, by doing more investigation on the application domain, or by looking at existing similar systems. The produced Formally Grounded specification should reflect the System where all these points have been settled.

The original use case based requirement specification should then be *revised* so as to be coherent with the Formally Grounded one. In general use cases and/or scenarios may be added or removed, scenarios may be modified by adding/removing steps or by making more precise the terminology used to describe them. In this way *the final outcome of our method will be not only a better and more systematic understanding of the System reflected in a Formally Grounded specification of the requirements, but also a more precise and sound use case based specification of the same requirements.*

Clearly task 5 will be performed in parallel with tasks 3 and 4.

4 The Auction System Case Study

In this paper we present a part of the application of our method to the case study of an Auction System proposed in [13] by S. Sendall; the remaining parts are in [5]. The description of the problem (from [13]) solved by the Auction System is shown in Fig. 1.

> Your team has been given the responsibility to develop an online auction system that allows people to negotiate over the buying and selling of goods in the form of English-style auctions (over the Internet). The company owners want to rival the Internet auctioning sites, such as, eBay, and uBid. The innovation with this system is that it guarantees bid placed are solvent, making for a more serious marketplace.
>
> All potential users of the system must first enroll with the system; once enrolled they have to log on to the system for each session. Then, they are able to sell, buy, or browse the auctions available on the system. Customers have credit with the system that is used as security on each and every bid. Customers can increase their credit by asking the system to debit a certain amount from their credit card.
>
> A customer that wishes to sell initiates an auction by informing the system of the goods to auction with the minimum bid price and reserve price for the goods, the start period of the auction, and the duration of the auction, e.g., 30 days. The seller has the right to cancel the auction as long as the auction's start date has not been passed, i.e., the auction has not already started.
>
> Customers that wish to follow an auction must first join the auction. Note that it is only possible to join an active auction. Once a customer has joined the auction, (s)he may make a bid, or post a message on the auction's bulletin board (visible to the seller and all customers who are currently participants in the auction). A bid is valid if it is over the minimum bid increment, and if the bidder has sufficient funds, i.e., the customer's credit with the system is at least as high as the sum of all pending bids. Bidders are allowed to place their bids until the auction closes, and place bids across as many auctions as they please. Once an auction closes, the system calculates whether the highest bid meets the reserve price given by the seller, and if so, the system deposits the highest bid price minus the commission taken for the auction service into the credit of the seller (credit internal with the system).
>
> The auction system is highly concurrent–clients bidding against each other in parallel, and a client placing bids at different auctions and increasing his/her credit in parallel.

Fig. 1. Auction System Problem Description

4.1 Auction System Task 1 – Use Case Based Requirement Specification

Here we report the use case based specification of the requirement on the Auction System given following the method of [14] as found in [13]. The only difference with [13] is that we summarize the actors and the use cases by means of a UML use case diagram, see Fig. 2, below, showing also the "*include*" relationships among the use cases (depicted by dotted lines). In the following use case descriptions "**" means that the details about some aspects of the Auction System (e.g., data format or rules to follow to perform some activity) are given in an accompanying document, not present in [13] and thus not considered here. Here we do not detail the schema for the use case description followed in this example and presented in [14]. For lack of room, we present only the main use case buy item under auction, the other ones are in [5].

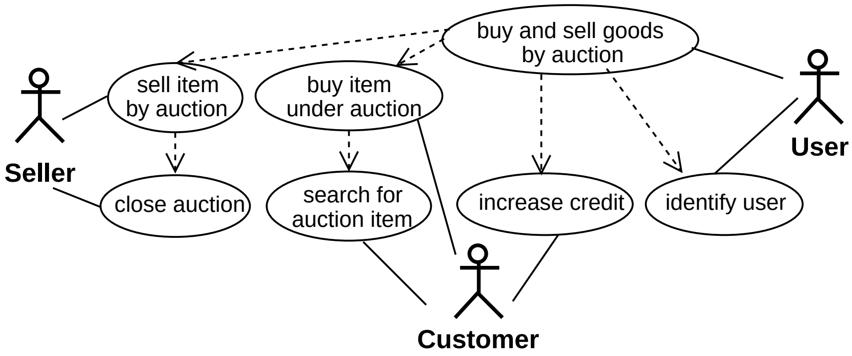


Fig. 2. Auction System: Use cases and actors

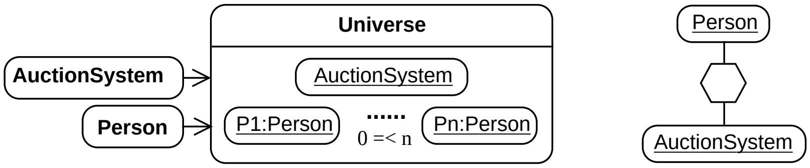
Use Case buy item under auction

Intention in Context: The intention of the Customer is to follow the auction, which may then evolve into an intention to buy an item by auction, i.e., (s)he may then choose to bid for an item. The Customer may bid in many different auctions at any one time. (Also the actor Participant means the Seller and all the Customers that are joined to the auction).
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System.
Main Success Scenario: Customer may leave the auction and come back again later to look at the progress of the auction, without effect on the auction; in this case, the Customer is required to join the auction again.
1. Customer searches for an item under auction (search item).
2. Customer requests System to join the auction of the item.
3. System presents a view of the auction** to Customer.
Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer
4. Customer makes a bid on the item to System.
5. System validates the bid, records it, secures the bid amount from Customer's credit**, releases the security on the previous high bidder's credit (only when there was a previous standing bid), informs Participants of new high bid, and updates the view of the auction for the item** with new high bid to all Customers that are joined to the auction.
Customer has the high bid for the auction.
6. System closes the auction with a winning bid by Customer.
Extensions:
2a. Customer requests System not to pursue item further:
  2a.1. System permits Customer to choose another auction, or go back to an earlier point in the selection process; use case continues at step 2.
3a. System informs Customer that auction has not started: use case ends in failure.
3b. System informs Customer that auction is closed: use case ends in failure.
4a. Customer leaves auction:
  4a.1a. System ascertains that Customer has high bid in auction:
    4a.1a.1. System continues auction without effect; use case continues at step 6
  4a.1b. System ascertains that Customer does not have high bid in auction: use case ends in failure.
4||a. Customer requests System to post a message to auction and provides the message content**.
  4||a.1. System informs all Participants of message; use case continues from where it was interrupted.

5a. System determines that bid does not meet the minimum increment**:
  5a.1. System informs Customer; use cases continues at step 4.
5b. System determines that Customer does not have sufficient credit to guarantee bid:
  5b.1. System informs Customer; use cases continues at step 4.
6a. Customer was not the highest bidder:
  6a.1. System closes the auction; use case ends in failure.

4.2 Auction System: Task 2

The Auction System has any number of context entities all of the type Person (anyone accessing the system by Internet). A Person may play three roles: User (plain Internet user), Customer (a User identified by the Auction System and connected with it) and Seller (a Customer selling some goods using the Auction System). We give a first version of the Context View showing the Auction System and the Persons (the incomplete cooperation diagram just shows that the a Person interacts only with the Auction System).



4.3 Auction System: Task 3

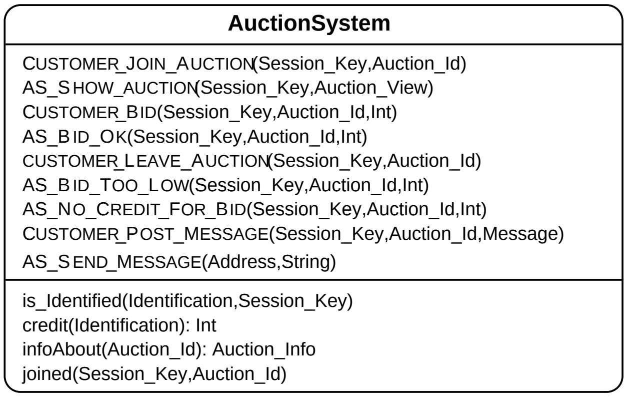
We examine the various use cases, one after the other, looking for the elementary actions and the state observers of AuctionSystem, together with the needed data structures and, possibly new context entities. Note that, for each use case, we do not give the features used by the included sub-use cases.

We name each elementary interaction made by the Auction System with an identifier of the form AS_. . . , whereas those made by a person context entity will be named User_. . . , Customer_. . . and Seller_. . . , depending on the role.

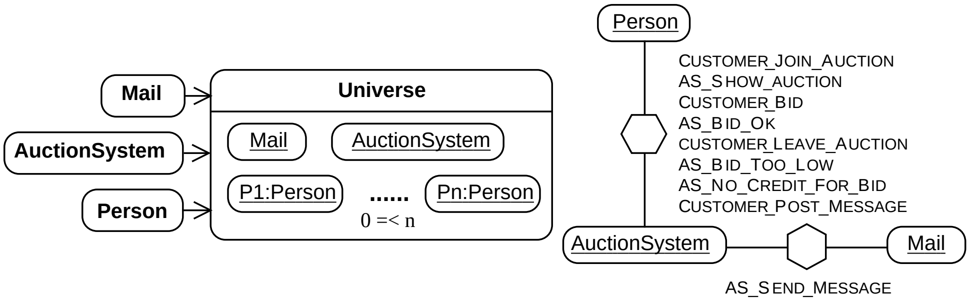
For each use case we produce a fragment of the Context View, of the Data View and of the specification of the Auction System. At the end, all these fragments will be put together getting the initial view of the structural part of the Formally Grounded requirement specification of the Auction System. To be able to support the evolution of the requirements, however, we require to keep track of the features of the specification (elementary interactions, state observers, and data structures) that are related with each use case.

Already, during this task many questions about the Auction System may arise that should be settled with the client; we use the following annotation for these questions and the way chosen to settle them **Q:** *problem* **A:** *settled in this way.*

Use Case buy item under auction The elementary interactions of AuctionSystem, shown in the first compartment of the above diagram, correspond either to an interaction made in the use case by the Auction System towards a context entity (e.g., AS_Bid_Ok for communicating that the placed bid was ok) or to an interaction received by a context entity (e.g., Customer_Bid for a Customer placing a bid). Instead, the state observers, in the second compartment, correspond to information recorded inside the Auction System either tested or updated during the use case (e.g., *credit*: the actual credit of a Customer denoted by an identification; *infoAbout*: the current information about an auction).

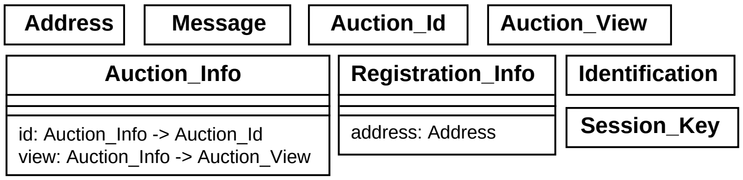

AuctionSystem
CUSTOMER_JOIN_AUCTION(Session_Key,Auction_Id)
AS_SHOW_AUCTION(Session_Key,Auction_View)
CUSTOMER_BID(Session_Key,Auction_Id,Int)
AS_BID_OK(Session_Key,Auction_Id,Int)
CUSTOMER_LEAVE_AUCTION(Session_Key,Auction_Id)
AS_BID_TOO_LOW(Session_Key,Auction_Id,Int)
AS_NO_CREDIT_FOR_BID(Session_Key,Auction_Id,Int)
CUSTOMER_POST_MESSAGE(Session_Key,Auction_Id,Message)
AS_SEND_MESSAGE(Address,String)
is_Identified(Identification,Session_Key)
credit(Identification): Int
infoAbout(Auction_Id): Auction_Info
joined(Session_Key,Auction_Id)

The Context View, see below, shows which context entities take part in the use case (e.g., the person), and which are the interactions of the Auction System with them (e.g., CUSTOMER_JOIN_AUCTION is an interaction between Auction System and the person).



Q: *This use case requires that the Auction System informs the participants to an auction about various facts (e.g., when there is a new higher bid or a message of another participant), but nothing is said on how that will be performed. In the description of the use case close auction there is a note saying that this is an open issue and that it will be likely made by email.* **A:** *It is assumed the existence of an external mail service, not further detailed, able to deliver messages to User identified by some kind of address (because the client will decide in future among email, SMS, messaging systems). The mail service will be then a NEW context entity (and a new secondary actor).*

The Data View shows all the data used as parameters by the found elementary interactions and state observers, and which predicates/operations we need to perform all the calculations over them required by the use case. For example, **Auction_Info**, the information about an auction, has an operation, `view`, for recovering a view of the auction to be shown to its participants, whereas **Auction_View** is not further detailed.



4.4 Auction System: Task 4

This task consists in finding the properties about the Auction System by filling the tableau generated by the elementary interactions and state observers found in the previous task, and by completing the specifications of the data structures. Clearly, while doing this activity, new state observers may be added, which will have then to be introduced in the tableaux and considered while looking for the properties. The original use case based specification may be modified by reflecting the better insights on the Auction System gained while looking for properties.

Here we show only some properties, together witht he arisen questions, about a few elementary interactions and state observers needed for the use case buy item under auction; each property is both expressed in our notation, and accompanied by a comment. The full set of the properties can be found in [5].

Elementary interaction Customer_Join_Auction Looking for the pre/postconditions of Customer_Join_Auction for filling the tableau cell whose both indexes are that elementary interaction, we found the following unclear points about the Auction System.

Q: *Does the use case* search item *ends having selected one auction or one item? This is relevant because there may be many different auctions for the same item, e.g., a used car. The description of* search item *suggests some auctions, whereas that of* buy item under auction *suggests one item.* **A:** *The* search item *ends with some selected auctions, as in other auction systems.*

Q: *Can an auction selected by the* search item *be in any status (e.g., closed or not yet started)?* **A:** *Yes, and this is quite sensible, since a Customer may be interested in knowing that some item has been sold in the past and at which price, or which are the current starting prices of some items, or that some items will be soon auctioned.*

Q: *Can a Customer try to join a closed or not-started auction?* **A:** *No, the Auction System should not provide this possibility, and answers with an error.*

The above problems lead us to revise the use case search item. As a result, we now have the *NEW* browse auctions use case ending with a selected group of auctions. Moreover, the use case buy item under auction may start only when there is one selected auction that is active. Then, we introduce a new state observer *selected_Auctions* that associates with each identified Customer (referred to by a session key *sk*) the identities of the currently selected auctions.

Q: *Can a Customer join an auction to which (s)he is already joined?* **A:** *Yes, since there is no problem. A better choice may be that the Auction System sends a warning to Customer.*

```
If a Customer joins an auction, then
  (s)he is identified,
  Customer has selected one auction that is active;
  and after (the Customer has joined that auction, and
  the Auction System shows to her/him all the detail of the selected auction)
if Customer_Join_Auction($sk$,$aid$) happen then
  exists $id$:$Identification$ s.t. $is\_Identified(id,sk)$ and
    $status(infoAbout(aid)) = active$ and
    $joined^{\mathsf{nxt}}(sk,aid)$ and
    in any case next AS_Show_Auction($sk$,$view(infoAbout(aid))$) happen
```

Elementary interaction AS_BID_OK While looking for its postcondition which concerns also the future behaviour of the Auction System after having performed the elementary interaction we detected the following problem.
Q: *Is it true that a Customer joined to an auction is informed twice of each new bid, once by receiving a view of the auction with the new bid and once by some kind of message? Moreover, if a Customer places a bid, and after leaves the auction, will (s)he be ever informed of a new higher bid? More generally, which is the intended duration of an auction? a few hours when the participants bid many times, and continuously look at the current view of the auction? or several days, when the participants from time to time place their bids and look at the situation of the auction?* **A:** *The client decided that an auction handled by the Auction System should last a few hours with all participants logged on; thus there is no need to inform the joined customers and the seller of the various bids, because they continuously examine the current view of the auction that the Auction System keeps updated.*

```
If the Auction System informs a Customer that her/his bid is ok, then
  the Customer placed such bid,
  (s)he had sufficient credit, and the bid met the minimum increment; and after
    the bid is recorded,
    the amount is secured by the Customer credit,
    the security on the previous high bid is released (if any), and
    the updated auction view is sent to all the Customers joined to the auction.
```

```
if AS_BID_OK($sk,aid,i$) happen then
  in any case before CUSTOMER_BID($sk,aid,i$) happened and
  $i \leq credit(identityOf(sk))$ and
  $ibid\_Ok(infoAbout(aid),i)$ and
  $high\_Bidder(infoAbout^{nxt}(aid)) = identityOf(sk)$ and $high\_Bid(infoAbout^{nxt}(aid)) = i$
   and $credit^{nxt}(identityOf(sk)) = credit(identityOf(sk))$ - $i$ and
  (if is defined ($high\_Bidder(infoAbout(aid))$ ) then
    $credit^{nxt}(high\_Bidder(infoAbout(aid))) =$
    $credit(high\_Bidder(infoAbout(aid))) + high\_Bid(infoAbout(aid)))$ and
  for all $sk_1$:$Session\_Key$
    • if $joined(aid,sk_1)$ then AS_SHOW_AUCTION($sk_1,view(infoAbout(aid)))$
```

State Observer *credit* The first version of the property about the decreasing of the credit (part of the tableau cell indexed by *credit*:*credit*) based on what is written in the various use case descriptions is the following, and points out a problem.

```
If the credit of a Customer decreases, then the Customer made a bid in an auction.
if $credit^{nxt}(id) = credit(id)$ - $i$ and $i > 0$ then exists $sk$:$Session\_Key$, $aid$:$Auction\_Id$ s.t.
  AS_BID_OK($sk,aid,i$) happened and $is\_Identified(id,sk)$
```

Q: *It is true that a Customer using the Auction System only for selling items will be never able to collect her/his money? Moreover, can a buying Customer recover her/his money when (s)he is no more interested in buying?* **A:** *Yes; thus we have to add a NEW use case* decrease credit *for allowing a Customer to recover her/his credit.*

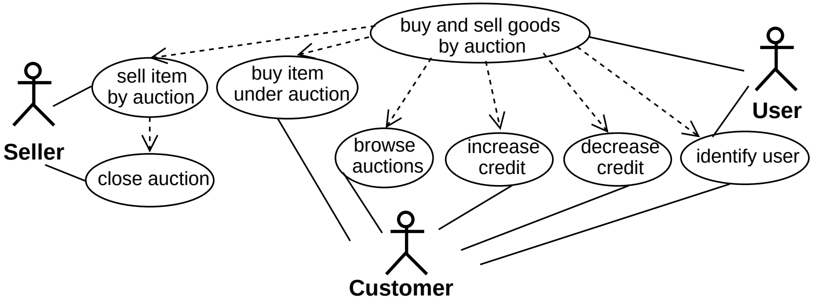
The new version we propose is then

```
If the credit of a Customer decreases, then
  either the Customer asked the Auction System to decrease it, (NEW)
  or the Customer made a bid in an auction.
```

```
if $credit^{nxt}(id) = credit(id) - i$ and $i > 0$ then
  exists $sk$:$Session\_Key$, $ctd\, Credit\_Transfer\_Detail$ s.t.
    AS_DECREASED_CREDIT($sk$,$ctd$) happened and
    $i = amount(ctd)$ and $is\_Identified(id,sk)$
  or exists $sk$:$Session\_Key$, $aid$:$Auction\_Id$ s.t.
    AS_BID_OK($sk$,$aid$,$i$) happened and $is\_Identified(id,sk)$
```

4.5 Auction System Task 5 – New Use Case Based Requirement Specification

Here we report only the new use case diagram and the new description of the use case buy item under auction, see [5] for the complete new use case based requirements. Two new use cases were identified when following our approach (see the previous section), browse auctions (thus, point 1. was removed from the buy item under auction description below) and decrease credit. The questions brought up by our work led to several modifications, e.g., the work on AS_BID_OK in Sect. 4.4 led to remove one part of point 5. in the new buy item under auction description below.



Use Case buy item under auction

Intention in Context: *UNCHANGED*
Primary Actor: Customer
Precondition: The Customer has already identified him/herself to the System
NEW: and selected one active auction.
Main Success Scenario: *UNCHANGED*
REMOVED: 1. Customer searches for an item under auction (search item).
2. Customer requests System to join the selected auction.
3. System presents a view of the auction** to Customer.
Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer
4. Customer makes a bid on the item to System.
5. System validates the bid, records it, secures the bid amount from Customer's credit**, releases the security on the previous high bidder's credit (only when there was a previous standing bid), *(REMOVED: informs Participants of new high bid,)* and updates the view of the auction for the item** with new high bid to all Customers that are joined to the auction. Customer has the high bid for the auction
6. System closes the auction with a winning bid by Customer.
Extensions:
UNCHANGED: 2a, 5a, 5b, 6a
3a. *NEW: The Customer is the Seller of the auction; System informs Customer that (s)he cannot join the auction. Use case ends with failure.*
REMOVED: 3a. System informs Customer that auction has not started: use case ends

in failure.

REMOVED: 3b. System informs Customer that auction is closed: use case ends in failure.

4a. Customer leaves auction:

4a.1a. System ascertains that Customer has high bid in auction:

4a.1a.1. System continues auction without effect; use case continues at step 5

4a.1b. System ascertains that Customer does not have high bid in auction: use case ends in failure.

4||a. Customer requests System to post a message to auction and provides the message content**.

4||a.1. *MODIFIED: System updates the view of the auction with the added message to all Customers that are joined to the auction;* use cases continues from where it was interrupted.

5 Conclusion and related works

In this paper we have proposed a method to review use case based requirements for a system by building a companion Formally Grounded specification. As a result the initial requirements are examined in a systematic way through the study of the various aspects of the considered system, modelled in terms of elementary interactions and state observers. For example, the possible interferences among different use cases may be revealed (elementary interactions relative to different use cases may yield a change of the same state observer), the communications between the system and the actors become more precise (they are modelled by elementary interactions, which require a precise definition of their parameters), the secondary actors (that help the system to satisfy the primary actors goals) are discovered and their features are clarified (all entities interacting with the system must be defined and modelled).

The produced Formally Grounded specification has a user-friendly notation (diagrams plus textual annotations in a natural-like language), and so it could be used as the requirement document. The proposed method also requires to update the original use case based requirements whenever a new aspect of the system is brought to light, thus, at the end, new improved use case based requirements are available. In the meantime, the formal CASL/CASL-LTL specification corresponding to the Formally Grounded one is also available, e.g., for formal analysis (but we have not yet investigated this point).

We think that starting to build directly the Formally Grounded specification from the description of the problem may be not as much as effective as the proposed combination of use cases and Formally Grounded specification, because the ingredients of the Formally Grounded specification (elementary interactions and state observers) are in some sense at a finer grain than the functionalities of the system, and so may be difficult to find by just considering the problem.

As an example, we have used our method on a medium-size case study, an electronic auction system. For lack of room, we have described here only parts of the various tasks and shown only some fragments of the produced artifacts; the complete development and the resulting artifacts can be found in [5]. The advantages shown by our method on this case study seem quite positive. Indeed, we have detected many problematic or not completely clarified aspects in the original use case based requirements. Among them, we recall (i) explicit auctions browsing functionality (blurred in the initial requirements: the information on all auctions were available but not shown), (ii) the fact that the auctions should be performed in a chat-like way, (iii) discovered the need for a decrease-credit functionality, (iv) made explicit that when a Customer unregisters any left credit goes to the Auction System owner.

Moreover, we would like point out that we did not write the starting use case requirements (given by Sendall[13] who, as of now, has no relationship with our group and our method), and we found them quite accurate, presented using a well-organized template and produced following a good method.

Concerning the possibility to use effectively the proposed method we would like to make the following positive points.
- It is possible, using common existing technologies, to build software tools to support the construction of the Formally Grounded specification, not only a graphical editor, but also wizards guiding the properties search.
- Each use case is linked with the elementary interactions, the state observers and the data structures used for its specification. This, together with the precise structure of the properties, may also help to support the evolution of the initial requirements; indeed a modification in one use case may be only reflected in a precise part of the associated Formally Grounded specification.
- The inspection and revision of the requirements proposed by our method concern only the nature of the system to be developed, and does not require to make any choice about the technology and methods that will be used to realize the system; thus it may be used in combination with many different methods.

In the literature there are other approaches to build a formal specification of the requirement of a system, but in general they do not aim at producing an improved non-formal specification. Among them, we recall the nice work of A. van Lamsweerde and his group[18], which offers a way to formally specify goal-oriented requirement specifications, and then to analyze them by means of formal techniques. R. Dromey[9] proposes to use "Behaviour Tree", a formal-visual notation to specify the requirements, then the resulting requirement specification will be used to derive the architectural structure of the system. Our approach, in the line of the well-founded methods [4], uses the underlying formal foundation to get a rigorous method to precisely specify the requirements, with the aim of achieving a careful inspection and a kind of validation of those requirements.

One of the authors, together with E. Astesiano, proposed another use case based method for the precise specification of the requirements [3], but using the (non-formal) UML statecharts as a notation to describe the use cases. However, because it does not offer a systematic way to analysis the System under different viewpoints, some aspects of the System captured by our method may not come under light.

We would like also to quote the work by S. Sendall and A. Strohmeier [15, 16] who promote the use of operation schemas (pre- and postconditions written in OCL) and system interface protocols (UML state diagrams) to complement use cases; our goal is different, that is to improve the use case based requirements.

Inspection techniques for improving the quality of a requirement specification (quite popular in Software Engineering practice, see e.g., [1]) are either based on ad hoc techniques or on check-lists. The main differences with our approach is that our "inspection" based on the underlying formal specification and the tableau-filling technique leads to a more systematic and precise examination of the requirements, whereas standard techniques lead to more generic checking. For instance, compare
"find and list all the ways the *credit* state observer may be updated in the various scenarios of all use case" (which helped to discover the lacking functionality of credit

decreasing), with
"Is there any missing functionality, that is, do the actors have goals that must be fulfilled, but that have not been described in use cases?" taken from [1]'s check-list.

References

1. B. Anda and D. Sjoberg. Towards an Inspection Technique for Use Case Models. In *Proc. SEKE 2002*. ACM Press, 2002.
2. E. Astesiano and G. Reggio. Labelled Transition Logic: An Outline. *Acta Informatica*, 37(11-12):831–879, 2001.
3. E. Astesiano and G. Reggio. Tight Structuring for Precise UML-based Requirement Specifications. In *Proc. of Monterey Workshop 2002: Radical Innovations of Software and Systems Engineering in the Future. Venice - Italy.*, LNCS. Springer Verlag, 2003. To appear.
4. E. Astesiano, G. Reggio, and M. Cerioli. From Formal Techniques to Well-Founded Software Development Methods. In *Proc. of The 10th Anniversary Colloquium of the UNU/IIST: Formal Methods at the Crossroads from Panacea to Foundational Support. Lisbon - Portugal, 2002.*, LNCS. Springer Verlag, 2003. To appear.
5. C. Choppy and G. Reggio. Improving Use Case Based Requirements Using Formally Grounded Specifications (Complete Version). Technical Report DISI-TR-03-45, DISI – Università di Genova, Italy, 2003. Available at ftp://ftp.disi.unige.it/person/ReggioG/ChoppyReggio03c.ps and pdf.
6. C. Choppy and G. Reggio. Towards a Formally Grounded Software Development Method. Technical Report DISI–TR–03–35, DISI, Università di Genova, Italy, 2003.
7. A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.
8. G. Costa and G. Reggio. Specification of Abstract Dynamic Data Types: A Temporal Logic Approach. *T.C.S.*, 173(2):513–554, 1997.
9. R. Dromey. From Requirements to Design: Formalizing the Key Steps. In *Proc. of SEFM'03, Brisbane - Australia*. IEEE Computer Society, 2003.
10. I. Jacobson, M. Christerson, P. Jonnson, and G. Overgaard. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison-Wesley, 1992.
11. P. Mosses, editor. *CASL, The Common Algebraic Specification Language - Reference Manual*. Lecture Notes in Computer Science. Springer-Verlag, 2003. To appear. Available at http://www.cofi.info/CASL_RefManual_DRAFT.pdf.
12. G. Reggio, E. Astesiano, and C. Choppy. Casl-Ltl : A Casl Extension for Dynamic Reactive Systems Version 1.0– Summary. Technical Report DISI-TR-03-36, DISI – Università di Genova, Italy, 2003.
13. S. Sendall. Case studies for RE_A2 course "Requirements Analysis with Use Cases". http://lglwww.epfl.ch/research/use_cases/RE-A2-case-studies/index.html, 2001.
14. S. Sendall and A.Strohmeier. Requirements Analysis with Use Cases. http://lglwww.epfl.ch/research/use_cases/RE-A2-theory.pdf, 2001.
15. S.Sendall and A.Strohmeier. From Use Cases to System Operation Specifications. In S. K. A. Evans and B. Selic, editors, *Proc. UML'2000*, number 1939 in Lecture Notes in Computer Science, pages 1–15. Springer Verlag, 2000.
16. S.Sendall and A.Strohmeier. Specifying Concurrent System Behavior and Timing constraints using OCL and UML. In M. Gogolla and C. Kobryn, editors, *Proc. UML'2001*, number 2185 in LNCS, pages 391–405. Springer Verlag.
17. UML Revision Task Force. *OMG UML Specification 1.3*, 2000. Available at http://www.omg.org/docs/formal/00-03-01.pdf.
18. A. van Lamsweerde. Building Formal Requirements Models for Reliable Software (Invited paper). In *6th International Conference on Reliable Software Technologies, Ada-Europe 2001*, number 2043 in Lecture Notes in Computer Science. Springer Verlag, 2001.