JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 2, March-April 2003

Specifying Use Case Interaction: Types of Alternative Courses

Pierre Metz and John O'Brien, Dept. of Mathematics & Computing, Cork Institute of Technology, Ireland

Wolfgang Weber, Dept. of Computer Science, Darmstadt University of Applied Sciences, Germany

Abstract

Use cases are a powerful and widely recognised tool for the elicitation and specification of functional software requirements. However, major problems and gaps still exist; practitioners frequently encounter these. One of these is the specification of alternative use case interaction courses. Experience shows that practitioners do not only need to specify alternative interaction courses that are inserted subject to a business condition; they also need to express partially or fully parallel interaction courses, exceptional use case behaviour, and cyclic interaction paths. Based on an extensive literature review and practical observations, this paper provides definitions for types of alternative interaction courses, as well as clarifying conceptual differences between, and providing illustrative real-world examples of, each. Moreover, these definitions are related to Cockburn's relevant practical approach of use case goals and use case business results in the context of goal-driven requirements engineering. Finally, the provided definitions will contribute to an understanding of use case interaction specification and goal-driven requirements engineering in practice; they also present clear advice on how to perform use case model refactoring through the application of UML's repeatedly discussed extend-relationship.

1 BRIEF HISTORY OF USE CASE BASICS

Goals and Interaction Courses

An *actor* specifies a role that can be taken by a person, a piece of hardware, a specific date/time, or a software component [4], [7]. [10], [11]. Each actor has certain operational responsibilities imposed by the business processes and business rules of the business domain. In order to fulfil its responsibilities, the actor has to perform a number of operations. An actor wants some subset of these operations to be facilitated by a software application or hardware apparatus. Thus, it sets corresponding goals for the system to deliver. These goals lead to desired system functionalities expressed by use cases [3]; each use case delivers a single goal.

A goal needs to be accomplished; therefore, some action has to be taken to achieve it. Hence, a use case goal leads to an interaction with a system in order to deliver this use case goal. The interaction description of a use case encompasses two parts [7]: a *Basic Course* describes the main sequence of interaction in which everything goes right ("happy path", see Example 1), whereas any non-frequent alternative or interruption from the basic course of interaction, such as optional parts of behaviour, alternative interaction parts, business error recovery and fault-handling, is called an *Alternative Course* [7] (see Example 1). In this sense, the term A*lternative Course* specifies a guarded variation of a part of another interaction course (see Figure 2) and, thus, is subject to some business condition. The location where an alternative course branches to another interaction course is called an *Extension Point*. It should be noted that, originally, UML introduced the term extension point to indicate branching locations for the extend-relationship only [2]; an extend-relationship is used to attach extracted optional behaviour, that resides in an additional use case, to its base use case. However, we consider the term extension point also applicable to branching points of alternative courses that remain local textual use case properties since this reduces the number of terms needed.

Example 1 shows an example of a basic course and an alternative that follows the specification technique in [3].

Example 1:

Consider a use case with the goal "Register New Student". Further consider the following fragment of the basic course:

```
1. Clerk enters new student's name, address, …
2. The system assigns a student ID.
3. Clerk assigns the university department.
4. System prints out a confirmation of
   registration.
5. …
```

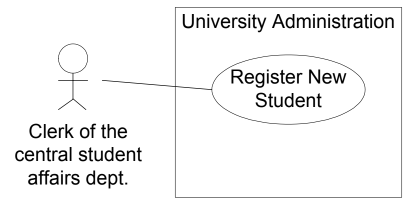
University Administration
Register New Student
Clerk of the central student affairs dept.

Figure 1: Use Case Diagram for Example 1

Now consider the following alternative course:

```
3a.  Student is a foreign student with pre-registered
     scholarship:

     3a1. The system shows the capacity of free campus
          appartments.
     3a2. The clerk assigns an appartment to the student
          ID.
     Rejoin at 3.
```

Scenarios vs. Interaction Courses

To fully comprehend the issues of use case business results and use case postconditions, the term scenario must be clearly understood.

A use case scenario is a single path through the use case's interaction courses. Figure 2 illustrates three interaction courses, i.e. one basic and two alternative courses. In fact, there are four scenarios: the sole basic course, basic course plus alternative course 1, the first part of the basic course and alternative course 2, and basic course with both alternative courses 1 and 2. Cockburn [3] refers to the basic course as the "main success scenario" because, if never branched by an alternative course, the basic course directly shows the use case goal succeeding.

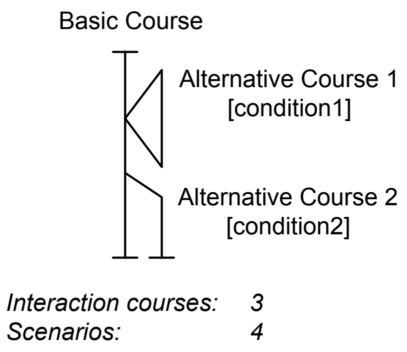

Figure 2: Illustration of the term "Scenario"

In Example 1, a possible scenario would be the sequence of the steps 1,2,3a1,3a2,3,4,5,...

Use Case Business Results and Use Case Pre- and Postconditions

The goal of each use case is associated with a use case interaction and also with a set of corresponding atomic business results. Since a use case has a business goal that should be guaranteed by following the use case interaction, a use case must have at least one measurable outcome that is of value to the business [3], [8], [9], [10]. Each of these quantifiable results is required by the business processes associated with the discussed use case; it is, also, delivered to at least one primary actor or stakeholder. Such a result is called a "use case business result".

> *Note that the business result is not to be mistaken with individual system responses to actor input while the interaction takes place. A use case business result specifies objectives that are expected to be delivered after the use case has finished[3],[8], [10], [14], [11].*

> *Further note that the explanation of a measurable result is vague and imprecise. However, it facilitates the finding of a correct and suitable level of specification detail [15]; this is one of the greatest problems in practice.*

Use case postconditions relate to goal achievement, i.e. to the existence and state of the delivered use case business results. Cockburn [3] refines the notion of use case postconditions: the fact that use case goals may be fully delivered, partially delivered or abandoned depending on failure recovery, i.e. alternative interaction courses, backup goals etc., leads to the specification of the *Minimal Guarantee* and a *Success Guarantee*.

The minimal guarantee specifies the least promise of a use case; for any failure scenario it is the result, while for any success scenario it is a part of the result [3]. The success guarantee represents additions to the minimal guarantee if the use case main goal is achieved, i.e. the full result of only the success scenarios [3]. The minimal guarantee is always implicitly included into the success guarantee. These two kinds of guarantees classify a use case's postconditions [11]. Therefore, since every use case goal has a business result, there is always a success guarantee for every use case. Consider the following guarantees for the use case given in Example 1:

Minimal Guarantee (always established, i.e. by both success and failure scenarios).

```
The system has logged all failures and the transaction date
and time.
```

Success Guarantee (established only by success scenarios):

```
The system has registered the student & Minimal Guarantee.
```

In many cases, a use case requires some condition to hold before it can be initiated and executed by an actor. These are called *Use Case Preconditions*. Use case preconditions are business-driven and can be derived from the surrounding business processes and underlying business rules; however, since a use case is the specification of a business task that is to be software-supported, use case preconditions are checked and guaranteed by the software application [1], [3], [5], [10], [11]. For example, the preconditions of a "Create Invoices" use case would include "At least one customer order is registered with the system".

Goals are Nested

An important property of goals is that they can be *nested*, i.e. they can have hierarchical dependencies [3]. Any goal has sub-goals which make up the super goal. In fact, any sentence in a use case interaction description can be considered a sub-goal of the overall use case goal. In Example 1 the assignment of a university department is a clear sub-goal of the registration of a new student which in turn is a sub-goal of the goal "Organise Courses of Study", for instance. As a consequence, super-goal and sub-goal relationships are transitive. Since goals lead to interaction, goal unfolding results in interaction refinement. In fact, any sentence in a use case interaction, i.e. a use case's sub-goal, could be further unfolded. Unfolding can only be done downwards, i.e. adding more detail. Conversely, each goal and its corresponding interaction can also be rolled up, i.e. procedural abstraction [13].

In Example 1, each interaction step of the basic course and the alternative courses are considered sub-goals of the use case goal "Register New Student".

Further Conclusions

It follows that an alternative course either reacts to a failure of an interaction step, i.e. to a failure of one of the overall use case's sub-goals, or it represents an actual interaction flow alternative. In both cases an alternative course sets a new sub-goal to the overall use case goal [3]. Consider the use case goal g_1 in Figure 3. Further consider a sub-goal $g_{1,1}$, i.e. an interaction step, of g_1. Now consider an alternative course setting a new sub-goal $g_{1,2}$ of g_1 that challenges $g_{1,1}$. This newly set sub-goal $g_{1,2}$ represents a backup goal or simply an alternative goal for the sub-goal $g_{1,1}$. This implies that both $g_{1,1}$ and $g_{1,2}$ reside at the same level because both are sub-goals of the overall use case goal g_1 (see Figure 3). We now recall that each alternative course includes a set of interaction steps, i.e. a set of sub-goals. Hence, the alternative course's goal $g_{1,2}$ is the super-goal of all the alternative course's interaction steps, i.e. of all subsequent sub-goals $g_{1,2,i}$ [3] (see Figure 3). However, the goals $g_{1,2,i}$ are also sub-goals of the overall use case goal g_1 because they are regular interaction steps of the overall use case and super-goal/sub-goal relationships are transitive.

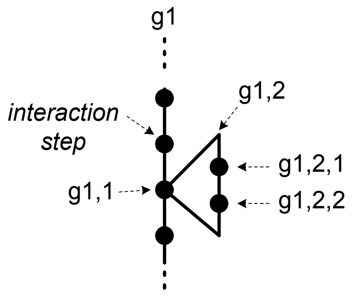

Figure 3: How goals, sub-goals and alternative courses are related

In Example 1, the use case's sub-goal of assigning the university appartment to the new student's ID in basic course at label 3 ($g_{1,1}$) may be postponed because the system detects that the applicant is a foreign student. On this condition, the additional goal of alternative course 3a "Assign Campus Appartment" ($g_{1,2}$) is addressed and, thus, the interaction steps 3a1 and 3a2 ($g_{1,2,i}$) are taken. After that, when the scenario has left the alternative course, the formerly postponed sub-goal at label 3 is carried out. However, both the goals of basic course at label 3 ($g_{1,1}$) and the alternative course 3a ($g_{1,2}$) reside at the same level of abstraction and are sub-goals of "Register New Student" (g_1).

As stated in above, each use case goal is associated with a use case interaction and also with a set of corresponding atomic business results. Since goals are nested, we can conclude that this is so for any goal at any level. As a consequence, each interaction step, i.e. each sub-goal, has business results; these can correspondingly be considered as "sub"-business results of the overall use case business results. More specifically, we may even consider each interaction step, i.e. each goal, at any level having a minimal guarantee and a success guarantee [3]. As a further consequence, an alternative course can also have guarantees because it also sets a sub-goal for the overall use case.

Note that Cockburn mentions that each sub-goal can be considered a "sub-use case" [3]. Cockburn also mentions that an alternative course can be considered a "miniature use case" [3]. This shows that it is possible to consider each goal at any level also having a minimal and a success guarantee. However, this must not be understood as a justification for arbitrarily splitting a use case because applying functional decomposition to use case models is highly disadvantageous.

Having reviewed general use case aspects and their implications, an examination of possible types of alternative courses, the definitions of same, and the relating of these to goals and goal-driven business results can be undertaken. This examination begins with a review of the current literature; solution proposals are presented subsequently.

2 TYPES OF ALTERNATIVE COURSES

What other authors say

According to current literature, the term *alternative course* is considered a general term that can be divided into three subtypes as follows:

1. **Conditional Insertion**
   Originally, conditional insertion is mentioned by Jacobson et al. in [7]. UML considers the semantics of the use case relationship «extend» as conditional insertion [1], [2], [14], [15], [16]. Conditional insertion is further referenced in

[1], [3], [10], [11], [14], [15], [16]. Conditional insertion means an optional part of behaviour that is subject to pure insertion into another course of interaction (see Figure 4), i.e. branching and returning to the branching point. Conditional insertion is not replacement; rather, it is a guarded addition of use case interaction. If the underlying condition is evaluated to true, the scenario running through will include this alternative course.

2. **Alternative History**
   As observed by Simons [14], an alternative course may also be viewed as an alternative history, i.e. a fully "parallel timeline". Here, timeline refers to a full or partial conditional replacement of the basic course or another alternative course: once having branched the base sequence, an alternative history never rejoins the base sequence or any other sequence (see Figure 4). Hence, an alternative history has XOR branching semantics. This is obvious since a use case always has only one thread of control [12].

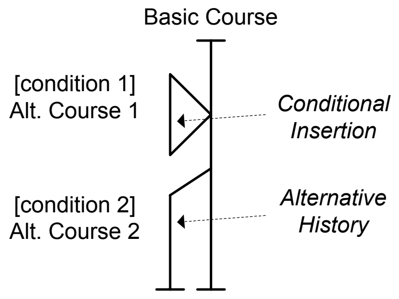

Figure 4: An alternative course may represent conditional insertion or an alternative history (following Simons [14])

Note that since an alternative history is meant in terms of location but not in terms of time, the term "history" for this type of alternative course remains confusing; however, the authors make use of this term to avoid increasing the quantity and complexity of terms by introducing an alternative new name.

3. **Use Case Exception**
   As observed in [1], [10], [14] an alternative course may further be viewed as a use case *Exception*. Simons [14] explains a use case exception as follows: after having branched the base sequence, a use case exception either returns to the end of the base sequence, or fully aborts the overall use case processing, i.e. never returns at all (see Figure 5). If it returns, then the rejoin point is never the same as the extension point. In Simons' view, the rejoin point of a use case exception is implicit and represents either the use case end or the branched alternative course's end. Also, like alternative histories and conditional insertions, a use case exception follows XOR branching semantics (see above).

   A comparison with programming languages' notion of exceptions reveals the following: the branching, i.e. the condition being evaluated to true, corresponds with the raising of a programming language exception. The interaction path of the use case exception corresponds with the recovery action of a programming language exception. Furthermore, a use case exception is always instantly caught after it has been raised and it can never be recast, whereas programming language exceptions may be left uncaught or may be recast.

Armour/Miller [1] explain a use case exception similarly to Simons with the following difference: in their view a use case exception may also return to the branching location of the base sequence, i.e. extension point and rejoin point can be the same.

All these views of a use case exception suggest that it can be handled and, thus, may still subsequently lead to a success scenario.

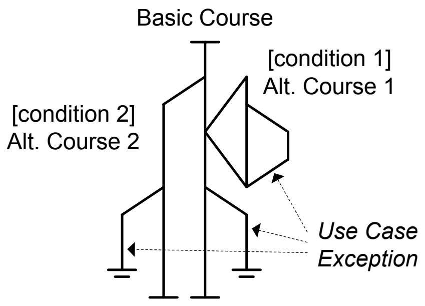

Figure 5: An alternative course may represent a use case exception (following [14])

Use case exceptions are not explicitly mentioned by Jacobson et al. in [7]. However, “errors” occurring during a use case execution are mentioned, but it is not clear that such a use case error can return to the base sequence. Use case exceptions are also mentioned by Kulak/Guiney in [10] but explained less precisely than by Simons [14].

Problems identified

A number of issues arise in connection with the above outlined views of alternative courses:

1. The notion of alternative histories and use case exceptions as explained by Simons [15] refers to the discussion of the semantics of UML’s extend-relationship, i.e. to the semantics of use case diagrams but not to use case narratives. Hence, the explanations in [15] are not directly related to the notions of use case goals and business-driven results. As a consequence, it is hard to find a conceptual difference between an alternative history and a use case exception in [15]: consider an alternative history and a use case exception both aborting to the end of the overall use case as depicted in Figure 4 and Figure 5. In this situation, both represent a variant of the basic course reaching the use case’s end; however, no further explicit semantic distinction is possible.

2. Although Jacobson et al. provide an explanation of alternative courses, our opinion is that these explanations are partially redundant and are also lacking in the clarity required to understand that alternative courses may characterise exceptions and alternatives. Furthermore, the work of Jacobson et al. is not expressed in the context of goal-driven requirements engineering.

3. Kulak/Guiney [10] consider use case exceptions as a practical need; they even suggest having corresponding individual items in the use case template. In their case studies several alternative courses can be found that compare with alternative histories or exceptions as defined above. However, they do not fully emphasise goal-driven functional requirements elicitation. Consequently, their explanation of use case exceptions is not related to goal-driven business results of use cases. Therefore, their distinction between a use case exception and an alternative course is less lucid.

4. Armour/Miller [1] use various terms for an alternative course without detailed explanation and observable distinction. These terms are “exception”, “alternative”, “variation”, and “extension”. There is a number of problems that arise in [1]: firstly, the way “alternative” and “exception” is used in the text leads to the impression that both terms are used synonymously. However, an exception is also explained as being allowed to rejoin, whereas an alternative is not. Secondly, it is stated that an exception may return to the branching point; in this case, it is hard to find a difference between their explanations of an exception and a guarded “insertion”. Furthermore, Armour/Miller’s approach is not goal-centric. Hence, the determination of the use case’s business results, i.e. postconditions, is not clearly related to the use case goal. As a consequence, their explanation of use case postconditions is not used to motivate different types of alternative courses. Moreover, the explanations of “alternative” and “exception” are given in the context of introducing UML’s semantics of the extend-relationship only; these terms are not related to the general semantics of use case interaction courses and their properties.

5. According to Cockburn in [3], an alternative course has an individual goal that leads to a success and a minimal guarantee as described above. Cockburn further discusses that an alternative course has an “end condition” ([3], p. 88) that may also be the achievement or the abandonment of the overall use case goal. This implies that an alternative course does not necessarily need to rejoin.

From all this we may infer that alternative histories and use case exceptions aborting to the end of the overall use case are supported; we may also conclude that alternative courses that do rejoin are still possible as illustrated by the examples used in [3]. However, different types of alternative courses are not introduced and explicitly mentioned. Moreover, in his use case template Cockburn does not suggest having individual items for individual types of alternative courses. Though existent it is difficult to clearly recognise these individual alternative course concepts in [3], so it may seem to the reader that this is not of major importance.

As a result, we see that there is no common and precise understanding of alternative courses. The discussion on alternative courses is not always related to the notions of goal-driven use case interaction and goal-driven use case results; this is so even though the use case concept can be related to an enterprise's business environment and customers wants and needs, an enterprise's economic strategic directions, its business processes and business rules in the context of goal-driven requirements engineering [3], [5], [6], [8], [14], [17]. These are problems practitioners frequently encounter when applying use cases in practice.

In the subsequent sections, we will give clear, condensed and integrated explanations of types of alternative courses. Improved definitions are provided that consider Cockburn's use case goals and goal-driven business results. Simons' and Armour/Miller's views of a use case exception having a rejoin point is reconsidered. Finally, significant clarification and organisation of the "terms jungle" associated with use case interaction courses should result.

3 "ALTERNATIVE HISTORY", "USE CASE EXCEPTION" AND "ALTERNATIVE PART" AS TYPES OF ALTERNATIVE COURSES

Definition of "Alternative History"

We share Simons' view that an alternative history remains a fully parallel timeline that never rejoins the branched master interaction course. The branched interaction course can be either the basic course or another alternative course but not a use case exception. A use case run, i.e. a concrete scenario, always ends after having followed an alternative history. As a consequence, an alternative history never has a rejoin point. Simons' initial description of an alternative history is extended by adopting Cockburn's view of an alternative course having a goal and guarantees: in addition to the basic course, an alternative history also always delivers the overall use case goal, i.e. it establishes the overall use case's minimal and success guarantee. This is indicated by the line going to the ground in Figure 6. Hence, all scenarios following an alternative history are success scenarios.

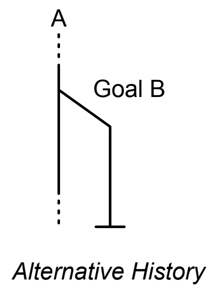

Figure 6: Graphical illustration of the term "alternative history"

We consider an alternative course with goal B as an alternative history if the following holds:

- B is a sub-goal of the super-goal A (either the overall use case goal or the goal of any other alternative course);
- No rejoin point is specified for the main interaction course of the alternative course with goal B;
- Success guarantee$_B$ $\Leftrightarrow$ success guarantee$_A$;
- Minimal guarantee$_B$ $\Leftrightarrow$ minimal guarantee$_A$;

The following example illustrates an alternative history:

Example 2:

Consider a use case with the goal "Register New Employee". Further consider the following basic course:

```
1.  The personnel manager enters new employee's name,
    address, and telephone number.
2.  The personnel manager assigns the company department.
3.  The system assigns a unique employee number.
4.  The system shows the scale of wages.
5.  The personnel manager determines the working hour rate.
6.  The personnel manager enters the weekly working hours.
```

Success Guarantee:

```
The system has registered the employee and has created a
payroll account & Minimal Guarantee.
```

Minimal Guarantee:

```
The system has logged all failures and the transaction date
and time.
```

Now consider the following alternative history:

```
4a.  The applicant will be paid outside the pay scale and
     will have flexible working hours:

     4a1. The personnel manager determines the new
          employee's
          monthly salary.
     4a2. The personnel manager determines fringe benefits.
          Use case ends successfully.
```

The applicant will be paid either by calculated wage, i.e. a worker, or by a fixed monthly salary, i.e. a manager. Since these two possibilities are mutually exclusive, the alternative course represents an alternative history. This case corresponds to Figure 6a.

Definition of "Use Case Exception"

In contrast to Simons [14], we consider a use case exception capable only, of aborting the entire use case (see Figure 7). A use case exception is not considered capable of aborting to the end of another alternative course. Hence, a use case exception never has a rejoin point. Furthermore, a use case exception is related to Cockburn's view of an alternative course having a goal and guarantees in the following way: a use case exception delivers only the overall use case's minimal guarantee; the overall use case's success guarantee is never established. This is indicated by the "earthed" line in Figure 7. The branched interaction course can be the use case's basic course or any other type of alternative course which is indicated by generally referring to goals A and B (see Figure 7).

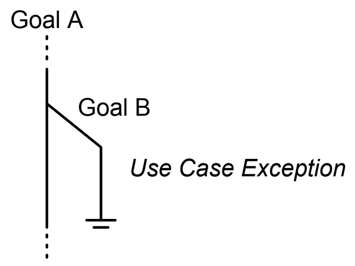

Figure 7: Graphical illustration of the term use case exception

We consider an alternative course with goal B as a use case exception if the following holds:

- B is a sub-goal of the super-goal A (either the overall use case goal or the goal of any other alternative course);
- No rejoin point is specified for the main interaction course of the alternative course with goal B;
- Success guarantee$_B$ $\Leftrightarrow$ minimal guarantee$_A$
- Minimal guarantee$_B$ $\Leftrightarrow$ minimal guarantee$_A$

This definition of a use case exception implies that a scenario following a use case exception is always a failure scenario. Once a scenario is following a use case exception, no alternative history can recover; a use case exception can never be rescued ("hell path"), i.e. the success guarantee can never be re-established (see Figure 8). A use case exception always aborts the entire system functionality the use case it belongs to is associated with, i.e. it brings the overall use case goal within the specified level of abstraction to fail.

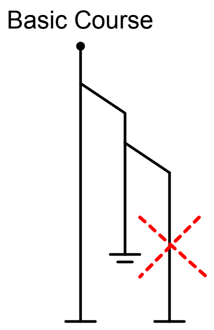

Figure 8: A use case exception cannot be branched by an alternative history that re-guarantees the use case's success guarantee

The following example illustrates a use case exception:

Example 3:

Consider a use case with the goal "Register New Customer Order". Further consider the following fragment of the basic course:

```
1. The sales clerk enters the customer's ID.
2. The system displays the customer's profile.
3. The sales clerk confirms that the customer's credit
   rating is sufficient and the order can be processed.
4. The system assigns an order ID.
5. The sales clerk registers the desired trade items.
6. …
```

Success Guarantee:

```
The system has initiated an order for the customer, has
documented payment information, and has registered the
request with the customer & Minimal Guarantee.
```

Minimal Guarantee:

```
The system has logged all failures and the transaction date
and time.
```

Now consider the following use case exception:

```
3a. The customer's outstanding debts are above the
threshold:

    3a1. The system notifies the key account manager
         responsible for mediation purposes.
    3a2. The system sets the order to pending status.
    3a3. The system informs the sales clerk.
    Use case aborts.
```

In the context of addressing the notion of use case exceptions it is essential to highlight the following: use case exceptions must not be mistaken for exception semantics of programming languages. In the programming domain, exceptions are unwanted but potentially expected errors that may occur at runtime. They may or may not be caught; however, for robustness they should. Further, they may or may not be recast. In contrast, in practice functional requirements of a prospective software application are derived from the enterprise's business environment, business processes, business rules, the enterprise's customers wants and needs, and the enterprise's economic strategic directions. In this respect, any kind of "business error" is considered an alternative path of a business process, i.e. in a business process there are no errors; rather, any business error and the reaction to it constitute a branch of the business process. For employees this is typically implemented in the form of a contingency plan.

As a consequence, a use case exception must be considered a special kind of alternative course that covers a software-supported alternative of a software-supported part of a business process. This clearly implies that a use case exception never reacts to some internal technical error (e.g., primary key violation in a database, division by zero, etc.) and that it is always "instantly caught" and never "recast"; otherwise, this would not be compatible with a business perspective. Consequently, since use case analysis and modelling treats the prospective system as a black-box, a use case specification, including its interaction descriptions, must be limited to the business language; therefore, a use case always remains a purely *descriptive* specification of an externally available system service [3], [5], [6], [10], [11].

Definition of "Alternative Part"

Practical experience shows that alternative histories and use case exceptions are not sufficient in order to support the needs of documenting functional requirements. In practice we need a richer set of concepts [1]. One such concept, conditional insertion (see Figure 9a), has already been identified by other authors (see above). However, conditional insertion is not sufficient. There is also a need to specify interaction cycles (see Figure 9b) and alternative interaction fragments (see Figure 9c). Hence, we recommend the introduction of another type of alternative course called an *Alternative Part*, the definition of which supports these three additional concepts.

We consider an alternative part as an alternative course that always returns to the base sequence that is being branched. Hence, an alternative part additionally specifies a *Rejoin point*. Figure 9 illustrates the above-identified alternative part types. Note that the base sequence can be the use case's basic course or any other type of alternative course which is indicated by referring to a super-goal A and a sub-goal B (see Figure 9).

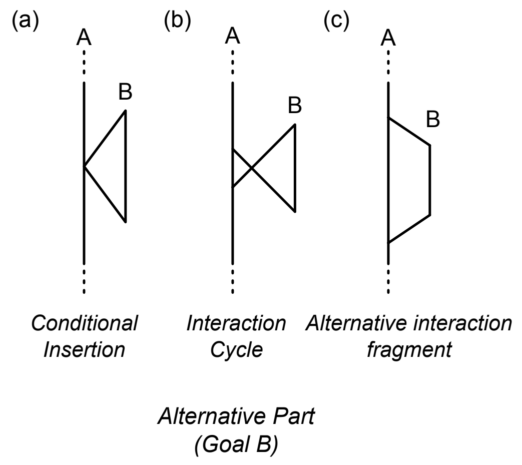

Figure 9: Illustration of the term alternative part

An example for conditional insertion (see Figure 9a) has already been introduced in Example 1. Consider the following examples for the cases shown in Figure 9b and in Figure 9c.

Example 4: Interaction cycle (see Figure 9b):

Consider the use case goal "Withdraw Cash" of an ATM.

Basic course:

```
1. The customer inserts the card.
2. The system validates that the card is valid.
3. The Customer enters the PIN.
4. The system validates that the PIN is correct.
5. …
```

Now consider the following interaction cycle and use case exception:

```
4a.  PIN has been entered incorrectly for the first or the
     second  time:

      4a1. The system logs the attempt.
      4a2. The system notifies the customer.
      Rejoin at 3.

4b.  PIN has been entered incorrectly the third time:

      4b1. The system logs the attempt.
      4b2. The system withholds the card.
      4b3. The system notifies the customer.
      Use case aborts.
```

Example 5: Alternative interaction fragment (see Figure 9c)

Consider the following altered basic course of the use case goal “Register New Employee“ from Example 2:

```
1. The personnel manager enters new employee’s name,
   address, and telephone number.
2. The system displays the vacancies and job category.
3. The personnel manager chooses the considered vacancy.
4. The system shows the scale of wages.
5. The personnel manager determines the hourly rate.
6. The system assigns an employee number.
7. The system assigns the company department based on the
   selected vacancy.
```

In this solution, the alternative history in Example 2 now becomes an alternative part , i.e. an alternative interaction fragment:

```
4a.   The applicant will be paid outside the pay scale and
      will have flexible working hours:

      4a1. The personnel manager determines the new
           employee’s monthly salary.
      4a2.  The personnel manager determines fringe benefits.
      Rejoin at 6.
```

We understand that an alternative part rejoins the branched interaction course before the branched interaction course has reached its end. This implies that after the rejoin point there is at least one possible interaction path within the overall use case delivering the overall use case goal, i.e. after having left an alternative part a scenario can still follow a success path establishing the overall use case's success guarantee. For alternative parts, however, we cannot impose restrictions on the relationship of the alternative part's guarantees (goal B) to the ones of the super-goal A.

A further essential requirement for alternative parts is that the extension point and the corresponding rejoin point always refer to the *same* base interaction course. Otherwise, the use case descriptions would follow harmful goto-/comefrom-semantics that have already been discouraged by Dijkstra since the beginning of the structured programming era as shown in [15]. Reintroducing such concepts to use case analysis, modelling, and specification would corrupt the major aims of, and advances in, requirements engineering as a part of modern software engineering methodologies.

These kinds of alternative parts do not need to be considered as individual types of alternative courses because their definitions do not differ. Furthermore, an alternative part also supports Simons' explanation of a use case exception being able to abort to the end of another alternative course (see above). It follows that a use case exception returning to the end of another alternative course as explained by Simons [15] represents an alternative part in our view.

4 CONCLUSIONS

The term „alternative course" of use case interaction descriptions has been clarified following rigorous examination and discussion of current literature, during which it was identified as a general term. Based on identified diversification, we have suggested the types *alternative history*, *use case exception*, and *alternative part*. Subsequently, we have provided definitions for these types. These definitions are related to Cockburn's practical approach of use case goals and use case guarantees [3] in the context of goal-driven requirements engineering [14]. Figure 10 graphically illustrates the relationships between the types of alternative courses; it also relates extension points and points of rejoin using UML syntax.

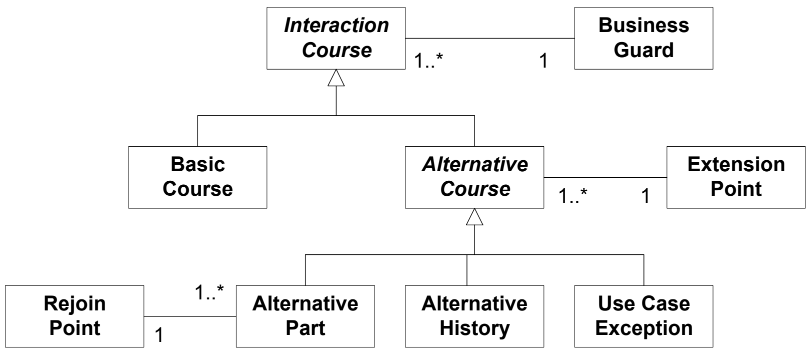

Figure 10: Graphical illustration of terms using UML syntax (no metamodel)

The suggested definitions reveal that the minimal guarantee of an alternative course, i.e. one of the use case's sub-goals, is always identical to the minimal guarantee of the overall use case; i.e. the use case's goal. This is common to all types of alternative courses. The reason for this is obvious: semantically, an alternative course always remains a part of the behavioural use case specification it is associated with. As a consequence, any alternative course can guarantee neither less nor more than the overall use case. The overall use case's minimal and success guarantee are relevant to all its scenarios [3].

Furthermore, the suggested definitions also have a substantial benefit for use case modelling with UML and use case model refactoring: in UML an extend-relationship specifies associated guarded use case behaviour. Hence, an extend-relationship can be applied in order to extract alternative courses into a new use case and to attach this new use case to the base use case [7]. In spite of the fact that doing so is possible, many software professionals disregard the use of the extend-relationship for extracting behaviour; rather, they prefer to keep alternative courses as local textual use case properties [3], [15]. However, an alternative course must still be extractable with an extend-relationship in order to remove redundancy from the use case model when this alternative course is common to at least two use cases. Whatever viewpoint is preferred, the definitions of the alternative course types presented are necessary in order to assist requirements analysts in determining the success and minimal guarantee of the newly created, and extend-attached use case holding this extracted alternative course. Unfortunately, the UML v1.4 metamodel supports conditional insertion only [1], [2], [15], [16]; any other kind of alternative course, as introduced above, is not provable in UML. Corresponding UML metamodel changes will be proposed in a future work.

The careful reader will have noticed that the extension point and the rejoin point in Example 1 refer to the same label. The reason is the following: in the current literature there is no commitment to include the interaction step referenced by the extension point into the scenario. Thus, the use case reader must decide on the semantic interpretation. These issues are discussed in a forthcoming paper where we will propose to always exclude the interaction step that is referenced by the extension point from the scenario; conversely, the interaction step that is referenced by the rejoin point is always *included* into the scenario. Extension points and points of rejoin are set correspondingly (see Example 1). With this approach, any type of alternative course can be handled and understood in a consistent manner; no interpretation, extra convention, UML stereotype or tag is needed. This enables a practitioner to focus on the business results only when documenting use case interaction, while preserving maximum freedom when designing an individual use case template. More detail will be provided in the forthcoming paper.

ACKNOWLEDGEMENTS

We wish to express our gratitude to the following individuals who have helped a great deal to improve this work. These are Birgit Friedrich and Shane Sendall. Special thanks to Roderick Coleman, Alistair Cockburn, Tony Simons, and Daryl Kulak. Furthermore, we thank all persons who published their ideas on requirements analysis and use case modelling on the internet from which the authors have gained benefit.

REFERENCES

[1] Armour F., Miller G. *Advanced Use Case Modeling*, Addison-Wesley, 2001

[2] OMG Unified Modeling Specification, Version 1.4, November 2000

[3] Cockburn A. *Writing Effective Use Cases*, Addison-Wesley, 2001

[4] Herzum P., Sims O. *Business Component Factory*, Wiley & Sons, 1999

[5] IBM Global Services, IBM Global Services Method Release 3.0, IBM Corporation, 1998, 2000

[6] IBM Object-Oriented Technology Center*, Developing Object-Oriented Software: An Experience-Based Approach*, Prentice Hall, 1997

[7] Jacobson I. Christerson M., Jonsson P., Övergaard, G. *Object Oriented Software Engineering – A Use Case Driven Approach*, Addison-Wesley, 1992

[8] Jacobson I., Griss M., Jonsson P. *Software Reuse - Architecture Process and Organization for Business Success*, Addison-Wesley, 1997

[9] Jacobson I. “The Road to the Unified Software Development Process”, Cambridge University Press, SIGS Reference Series, 2000 (compilation of formerly published articles of I. Jacobson, revised and updated by S. Bylund)

[10] Kulak D., Guiney E. *Use Cases – Requirements In Context*, Addison-Wesley, ACM Press, 2000

[11] Metz P., wibas Schulung und Beratung GmbH, *Requirements Engineering, training course for software professionals and project managers*, Germany, 2001, http://www.wibas.de

[12] Metz P., O’Brien J., Weber W. “Against Use Case Interleaving”, *UML 2001 – Modeling Languages, Concepts, and Tools*, eds. Gogolla M., Kobryn C., Proceedings of the 4th International Conference on the UML, Lecture Notes on Computer Science 2185, Springer Verlag, Germany, 2001

[13] Oxford Dictionary, 1986

[14] Paech, B. *Aufgabenorientierte Softwareentwicklung - Integrierte Gestaltung von Unternehmen, Arbeit und Software*, Springer Verlag, Germany, 2000

[15] Simons A. “Use Cases Considered Harmful”, *Proceedings of TOOLS-29 Europe*, eds. R Mitchell, A C Wills, J Bosch and B Meyer (Los Alamitos, CA: IEEE Computer Society, 1999), p. 194-203, available at http://www.dcs.shef.ac.uk/~ajhs/abstracts.html#harmful

[16] Simons A., Graham I. “30 Things That Go Wrong In Object Modelling With UML 1.3”, *Behavioral Specifications of Businesses and Systems*, eds. H. Kilov, B. Rumpe, I. Simmonds, Kluwer Academic Publishers, 1999, p. 237-257, available at http://www.dcs.shef.ac.uk/~ajhs/abstracts.html#uml30thg

[17] Sommerville I., Sawyer P. *Requirements Engineering – A good practice guide*, John Wiley & Sons, 1997

About the authors

Pierre Metz is a postgraduate PhD student at the Dept. of Mathematics and Computing, Cork Institute of Technology, Ireland. His is interested are in the field of development process frameworks and methods, CMMI and ISO 15504/SPICE, requirements engineering and quality assurance, use case analysis and modelling, object-oriented foundations, and UML. Pierre can be reached at pmetz@cit.ie

John O'Brien joined the lecturing staff of the Maths and Computing Department, Cork Institute of Techbnology, CIT in 1987. Here he has lectured on, and developed courses in, Software Engineering at undergraduate and post-graduate levels. His current research areas include quality software engineering and project management. John is a current member of the academic council at CIT.John can be contacted at jobrien@cit.ie

Wolfgang Weber is a full professor at the Dept. of Computer Science at Darmstadt University of Applied Sciences, Germany. He works and teaches in the field of project management, software engineering, UML, object-oriented methods and programming. Wolfgang can be reached at w.weber@fbi.fh-darmstadt.de