7 A PrOCEDURAL FRAME

"Arithmetic is where the answer is right and everything is nice
and you can look out of the window and see the blue sky
--or the answer is wrong and you have to start over
and try again and see how it comes out this time."

[Carl Sandburg, Arithmetic, Complete Poems, 1950]

7.1 Introduction

The translation principles described previously are included in this chapter in a procedural frame whose aim is to make systematic the elaboration of the integrated semi-formal/formal model of the system. Although given as a series of interconnected steps and although a "regular" sequence of steps is proposed, this procedural frame is intended only to guide the development of the model, and not to insist on a pre-established sequence of modelling activities. As shown in this chapter, the frame is flexible enough to accommodate various specification strategies and to support the iterative development of the model. The artefacts obtained in the modelling process are described, including a key modelling element, denoted class compound and introduced primarily for supporting the formalisation process. This new construct represents an extension of the fundamental concept of class and encompasses the traditional UML class and the UML state diagram associated with the class. The specific modelling activities are also described and comments on the various elaboration paths that can be followed during the development of the combined UML/Z++ model are included. In addition to the suggested "regular" sequence of modelling activities an example of an alternative scenario for the modelling process is given. The regular modelling scenario proposed in this chapter is applied on the case study described in Chapter 8 and the entire procedural frame is supported by the Harmony environment presented in Chapter 9.

7.2 Modelling Focus

As indicated in Section 3.1, the approach presented in this thesis is focused on the structural and behavioural aspects of TCS and is aimed at developing OO models in a rigorous, pragmatic, and efficient way. For this reason, a number of modelling activities supported by UML are not included in the procedural frame described in this chapter and their corresponding artefacts (specifically, diagrams) are not included in the integrated UML/Z++ model. This simplification is justified by the fact that the above diagrams are either parallel to some already incorporated (specifically, collaboration diagrams are essentially re-writings of sequence diagrams), can be ignored without losing significant insight into the system (activity diagrams), or can be deferred to later development stages that are beyond the scope of the approach proposed in this thesis (component diagrams and deployment diagrams).

By considering the 4+1 architectural views shown in Fig. 3.4, only the User View, the Structural View, and the Behavioural View are dealt with in the proposed approach, and from the diagrams that support them only the use case diagrams, the class diagrams, the sequence diagrams, and the state diagrams are employed. In addition to the discarded diagrams indicated in the previous paragraph, object diagrams are not utilised either, the reason being twofold: firstly, they bring relatively little information about the system in addition to that already contained in class diagrams and sequence diagrams ("object diagrams show instances instead of classes; they are useful for explaining small pieces of complicated relationships, especially recursive relationships" [TogetherSoft00a]), and, secondly, the objects do appear in sequence diagrams in a more important role, that of describing behaviour (the system structure being sufficiently expressed by classes).

In short, we look at a 2+1 views architecture of the system, a reduction of the generic 4+1 views approach that nevertheless allows a reliable description of the system. It is worth noting that many of the UML applications described in the recent literature focus typically on use cases, scenarios, class diagrams, and statecharts diagrams, e.g., [Howerton99, Barrios99, Xie99, Jigorea00, Xu00] and less frequently other types of diagrams are also presented, e.g.,

[Bell99, Fernandes00]. In fact, having the class organisation completed in terms of both attributes and operations allows the further development of the system possibly up to and including implementation (partition and deployment of components may or may not be necessary, depending on the application).

7.3 Artefacts

During the modelling of the system, a series of diagrams are drawn, modelling constructs are completed, including both UML and Z++ specification of classes, and the formalisation and deformalisation processes are performed. Starting from a set of requirements that describe the desired properties of the system, the following five categories of artefacts (products) are obtained, making up the combined semi-formal/formal model of the system:

- Use case diagrams, describing the intended high-level behaviour of the system as seen from the point of view of external entities (actors) that interact with the system. These are typical UML use case diagrams, each capturing a portion of the system's externally visible behaviour (its "functionality"), and each containing a number of use cases that further detail this behaviour. The regular UML notation is used to develop both use case diagrams and use cases. Abbreviations for these constructs, introduced for easier referencing and used as prefix denominations within Harmony's Project Pane described in Chapter 9 are UC for use cases and UCD for use case diagrams;
- Scenarios, specific sequences of actions involving the system and the actors that interact with it. Scenarios, as pointed out by Booch, "are to use cases what instances are to classes, meaning that a scenario is basically one instance of a use case" [Booch98, pp. 225]. UML provides sequence and collaboration diagrams for representing scenarios; however, these diagrams involve a high level of detail (they require the designation of classes and objects for carrying out the scenarios) so we felt necessary to introduce a distinction between a scenario and a sequence diagram. Specifically, we see a scenario as an informal, analysis-level description of a particular sequence of actions encompassed by a use case, while a sequence diagram is a detailed, design-level description of the same thing (in sequence

diagrams responsibilities for carrying out actions are assigned to individual classes and objects, as opposed to the system as a whole.) In our approach, another major difference between scenarios and sequence diagrams is that no specific notation is required to represent scenarios, while sequence diagrams are developed using the UML notation. From a development point of view, a refinement step is thus introduced between the elaboration of scenarios and that of sequence diagrams. Scenarios can be written in natural language, possibly as a series of numbered steps [Schach99], captured in decision tables [Davis93], shown using custom-made, application-specific graphical aids (Chapter 8 provides an example), or described in a notation similar to that of sequence diagrams (e.g., event traces [Rumbaugh91]). In order to provide a modality to relate scenarios with their encompassing use case, the notion of group of scenarios, a basic structuring mechanism, is introduced. The abbreviation associated to a scenario is SC and the one for a scenario group is SCG;

- Sequence diagrams, developed using the UML notation and providing a design-level representation of scenarios. As discussed above, they are also "instances of use cases" and capture the externally visible behaviour of the system but in addition they show internal interactions among objects. The abbreviation for sequence diagrams is SQD and, by symmetry with scenarios, the notion of group of sequence diagrams is introduced, with the associated abbreviation SQDG;
- Class diagrams, defining the high level architecture of the system and consisting of classes, relationships among classes, and additional structural constraints expressed as multiplicity values. For formalisation purposes, only UML classes and the usual types of relationships indicated in Section 6.3.1 are considered. Class diagrams are represented using their dedicated UML notations, and are abbreviated as CD;
- Class compounds, each class compound, denoted COMP, being a simple syntactic extension of class, grouping the regular class description (CLS) and the state diagram associated with the class (CLSTD). The notion of class compound is introduced primarily for supporting the needs of the formalisation process but it represents in general a simple yet useful extension of the concept of class. The idea of a class compound comes naturally from Z++, but it has also been inspired from the approach of

Howerton and Hinchey [Howerton99], who propose the annexation of the Z specification of the class state diagram to the UML description of the class. The intention of Howerton et Hinchey is to directly combine UML descriptions and Z specifications for describing classes in an approach that advocates different notations for modelling different aspects of the system. However, they do not envisage the syntactical concatenation of the UML class and state diagram constructs and do not propose a denotation for their solution. As a brief remark, it is only natural to add when necessary the state diagram of the class (defining possible sequences of executions) to the two traditional sets of elements encapsulated in a class: data (defining structure) and operations (defining behaviour). Thus, the class compound concept can be viewed as a class with enhanced description of behaviour. Of course, not all classes need a state diagram, so the CLSTD section of COMP can be empty. A summary of the abbreviations introduced above is given in Table 7.I.

- Z++ specification (ZSPEC), consisting of a set of Z++ classes (ZPPCs), each Z++ class corresponding to a class from the UML space. The Z++ specification as a whole is the formal counterpart of the combined contents of the class diagrams that make up the UML component of the integrated model of the system.

Table 7.I Abbreviations for Modelling Artefacts

Element	Abbreviation	Element	Abbreviation
Use Case	UC	Class Diagram	CD
Use Case Diagram	UCD	Class Compound	COMP
Scenario	SC	Class Description	CLS
Scenario Group	SCG	Class State Diagram	CLSTD
Sequence Diagram	SQD	Z++ Specification	ZSPEC
Sequence Diagram Group	SQDG	Z++ Class	ZPPC

7.4 Activities

Fig. 7.1 gives a diagrammatic description of the procedural frame proposed in this thesis for modelling TCS. The figure shows both the modelling activities (steps) performed and the artefacts obtained as the result of each activity. Since the artefacts have been discussed in the preceding section, the focus is here on the activities. Before discussing them, a number of conventions and simplifications used in Fig. 7.1 are indicated.

7.4.1 Conventions in the Diagrammatic Representation of the Procedural Frame

Several conventions are used in Fig. 7.1, as follows:

- Activities are represented by rounded rectangles;
- Modelling products (the artefacts) are represented by regular rectangles;
- Continuous, arrow-ended flow lines connect activities with their output products and products with activities that use them as input;
- Dashed, arrow-ended lines represent a change from an activity to another and, in contrast with the continuous flow lines, do not require that artefacts are obtained in the originating activity (the decision to move to another activity may be based on the inspection of the already existing artefacts associated with the current activity). These dashed lines are used in two situations: in the process of iterative development of the model (feedback links), and when moving from one activity to another without necessarily providing new input to the newly initiated activity;
- The steps are numbered and organised in five stages (or levels), their ordering suggesting the typical flow of activities within the modelling process. Same level activities can be performed in a parallel fashion, including an interleaved form of parallelism;
- The set of diagrams obtained as a result of a specific modelling activity in stages 1 to 3 are generically denoted collection, e.g. the Use Case Collection is created in the Definition of Use Cases step.

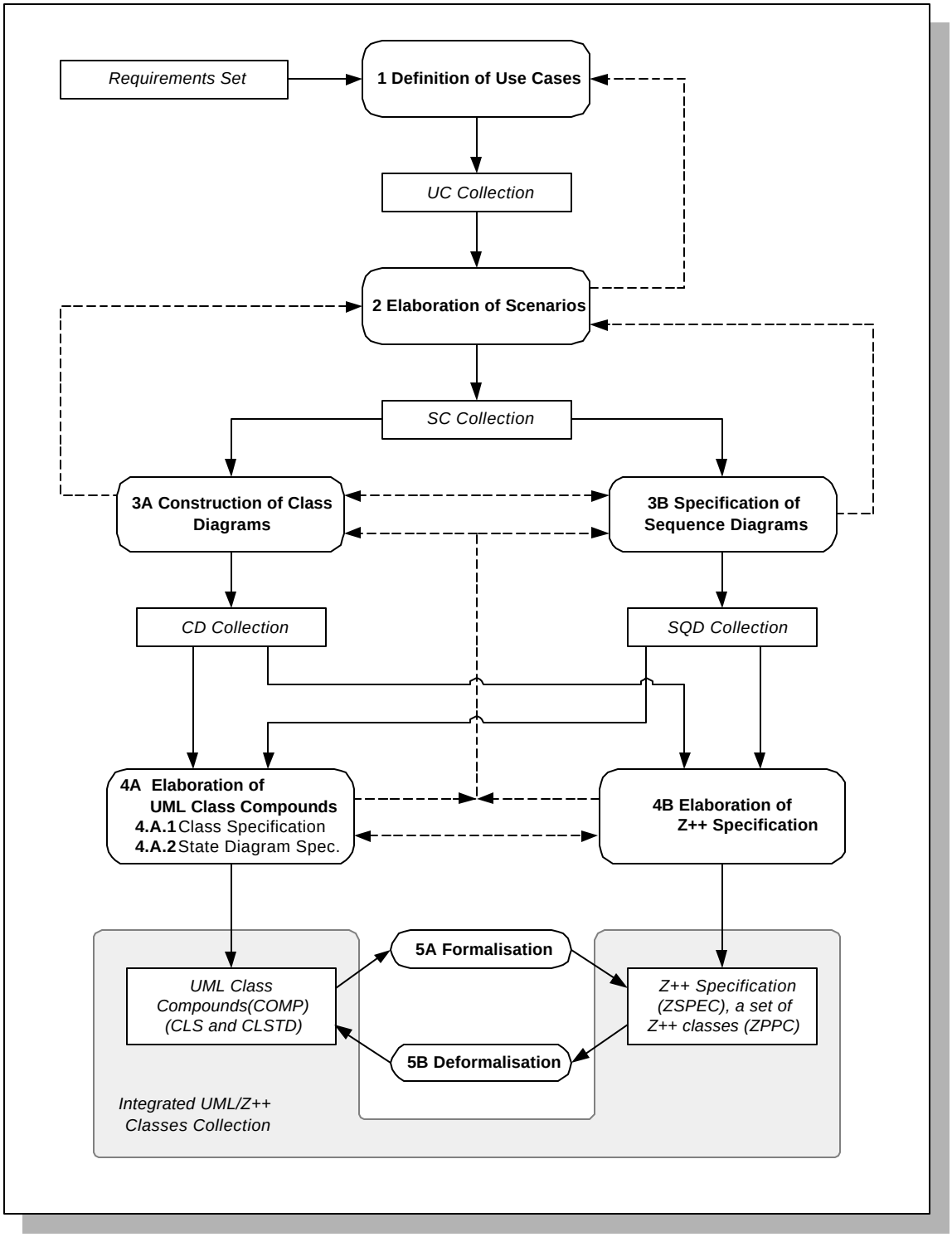

Fig. 7.1 The Procedural Frame

7.4.2 Simplifications in the Diagrammatic Representation of the Procedural Frame

In order to keep the diagram readable a number of simplifications have been made regarding aspects of the modelling process that are less common and therefore less emphasised in the proposed procedural frame. Firstly, while in principle it is possible to come back several levels at a time, for instance from step 4A to step 2 or even to step 1, the revision links are drawn however only from one level to the immediately preceding one (this would be the regular, more frequent way of refinement). Secondly, the CD Collection can serve as input for the Specification of Sequence Diagrams activity (3B) and, vice-versa, the SQD Collection can be used as reference for the Specification of Class Diagrams activity (3A), but more important is their respective input for activities 4B (Elaboration of UML Class Compounds) and, respectively, 4A (Elaboration of Z++ Classes). Thirdly, there is an implicit feedback from activities 5A (Formalisation) and 5B (Deformalisation) to activities 4A and 4B (in fact, so strong a feedback that activities on levels 4 and 5 can be aggregated on a single level, but we needed to highlight the steps of formalisation and deformalisation), which again is not shown for keeping the diagram readable. Lastly, the input for the entire process is represented by the Requirements Set, whose elaboration is not of our concern (we assume that a workable collection of requirements is available). Yet, in practice there is a continuous need for revising the requirements, so links back from modelling activities to the definition of requirements (an activity not shown in Fig. 7.1) should be considered implicit in the diagram.

7.4.3 Stages and Steps

The procedural frame outlined in Figure 7.1 serves only as a guide for modelling TCS, the most important thing being to correctly and completely develop all the artefacts of the integrated UML/Z++ model. The diagram presented in Fig. 7.1 is flexible enough to accommodate various specification strategies and encompasses diverse modelling paths, as discussed more in the next Subsection. In the following, we highlight the modelling activities included in our procedural approach and present them as organised in five stages. The

ordering of the modelling stages corresponds roughly to the typical sequence of activities so the discussion that follows is somewhat biased towards the "regular" modelling scenario proposed in the next Subsection and shown in Fig. 7.2. However, possible variations in the sequencing of activities are also indicated, and are further illustrated in Subsection 7.4.5.

The specific activities performed in each stage are the following:

- At stage 1, starting from the Requirements Set that describes the desired system, a number of use cases that capture segments of externally visible system functionality are identified, making up the Use Cases Collection of the integrated model. During this activity actors interacting with the system are also identified;
- At stage 2 use cases are used to instantiate a number of scenarios that will serve later for the identification of classes. There is no restriction on the way scenarios are represented since they are expected to produce "a rough cut" of the externally visible behaviour of the system and provide high-level insight into the application. Normal scenarios (most likely to occur) as well as abnormal scenarios (or exceptional scenarios, describing situations that diverge from the normal case) are developed and possibly tied together in a Scenario Group that corresponds to a particular use case. Taken together, groups of scenarios as well as individual scenarios (that is, scenarios not yet related to an already defined use case) make up the Scenarios Collection of the model. Although initially individual scenarios as well as groups of scenarios not bounded to uses cases are possible, it is recommended that through iterative revision of specifications the final model should contain only bounded groups of scenarios, a one-to-one correspondence use case-scenario group being desirable;
- At stage 3, using the available Scenarios Collection two possibly intertwined activities can take place: Specification of Class Diagrams (3A) and Specification of Sequence Diagrams (3B). In practice, one needs to develop concurrently the system's model on both directions, structure (classes) and behaviour (primarily, operations included in classes). Only by simultaneously considering the class structure and the responsibilities of classes and class instances, as captured in sequence diagrams, can the catch-22 type of problem

at this level be resolved (what classes and objects to include in the sequence diagrams if the class diagram is not defined, and what classes make up the high-level architecture of the system if the internal behaviour is not known? –recall that scenarios describe externally visible behaviour). However, in practice, the specification of sequence diagrams is the one that can be deferred since in general it is easier to construct the class diagrams by exploiting the information contained in scenarios (class diagrams may contain only the names of the classes, without any other details, while the sequence diagrams necessarily include both classes and their operations). Thus, step 3A is normally performed first and step 3B follows. In fact, the specification of sequence diagrams performed in step 3B can be omitted all together, as shown in Subsection 7.4.5. The best thing, however, is not to ignore it, but to use it at least as a "revision checkpoint," with input from all subsequent levels. In short, from our point of view, on stage 3 the development of class diagrams is compulsory while the development of sequence diagrams is recommended;

- At stage 4 the CD Collection as well as the SQD Collection (if available) provide the basis for the detailed specification of classes. An argument can be raised about the development of classes represented separately from the development of class diagrams and, indeed, there is a blurred line between these two activities. We separate them for systematisation purposes and view the Specification of Class Diagrams as an activity in which the rough sketch of the system's class structure is drawn (in terms of classes, relationships, and cardinality constraints) while the subsequent activities of UML and Z++ class elaboration are concerned with the specification of class details (attributes, operations, and constraints). And, indeed, stages 3 and 4 are the closest related stages in the "stratification" suggested in Fig. 7.1. Regarding the "parallel" steps 4A, Elaboration of UML Class Compounds, and 4B, Elaboration of Z++ Classes, they can be started and performed simultaneously (this is the reason for placing them on the same level) but the typical way is to perform step 4A first or to perform only the step 4A and rely on the subsequent formalisation of class compounds (step 5A) to obtain Z++ specifications of classes. In the regular flow of activities shown in Subsection 7.4.4 we actually use step 4B as a refinement activity, which follows step 5A. The Elaboration of UML Class

Compounds on stage 4 consists in: (a) establishing the attributes and the operations of the classes as well as the constraints attached to classes in the regular UML construct of class (CLS); and (b), in drawing state diagrams (CLSTD) for those classes that require them, thus completing for each class in CDs its corresponding class compound COMP;

- At stage 5 the formalisation of selected UML class compounds takes place in step 5A by applying initially the rules for automated translation described in Chapter 6 and then by manually adding the necessary details to the formal specification. This activity has the role of producing rigorous descriptions of the system, captured in the Z++ specification. It provides the strongest basis for refining the model, many ambiguities, omissions and inconsistencies being detected here. At the same level of modelling, deformalisation of classes initially written in Z++ (step 5B) can be performed according to the guidelines suggested in Chapter 6. However, as shown in the next Subsection, the "regular" flow of activities includes only step 5A at this level of modelling and the procedural frame treats the activity of deformalisation as a "variation" of the modelling process.

It is important to note that iterative refinements of the products obtained so far need be performed. We envisage essentially two categories of iterative development, one expectedly more intensive (more frequently performed), which can be called "short range revision" because it involves backwards stages 5 to 3 only, and the second performed less frequently yet reaching farther, which can be called "long range revision," potentially affecting backwards all the stages of activities, including the (not shown in the diagram) elaboration of requirements. As already mentioned the iterative development of the artefacts that constitute the integrated model of system is primarily propelled by the process of formalisation, a key idea of our approach being to use formalisation of UML constructs for improving the chances of detecting errors.

7.4.4 The Regular Sequence of Modelling Activities

A graph-like representation of the regular sequence of modelling activities, which for simplicity omits the products of each activity, is represented in Fig. 7.2. (In UML terms, this

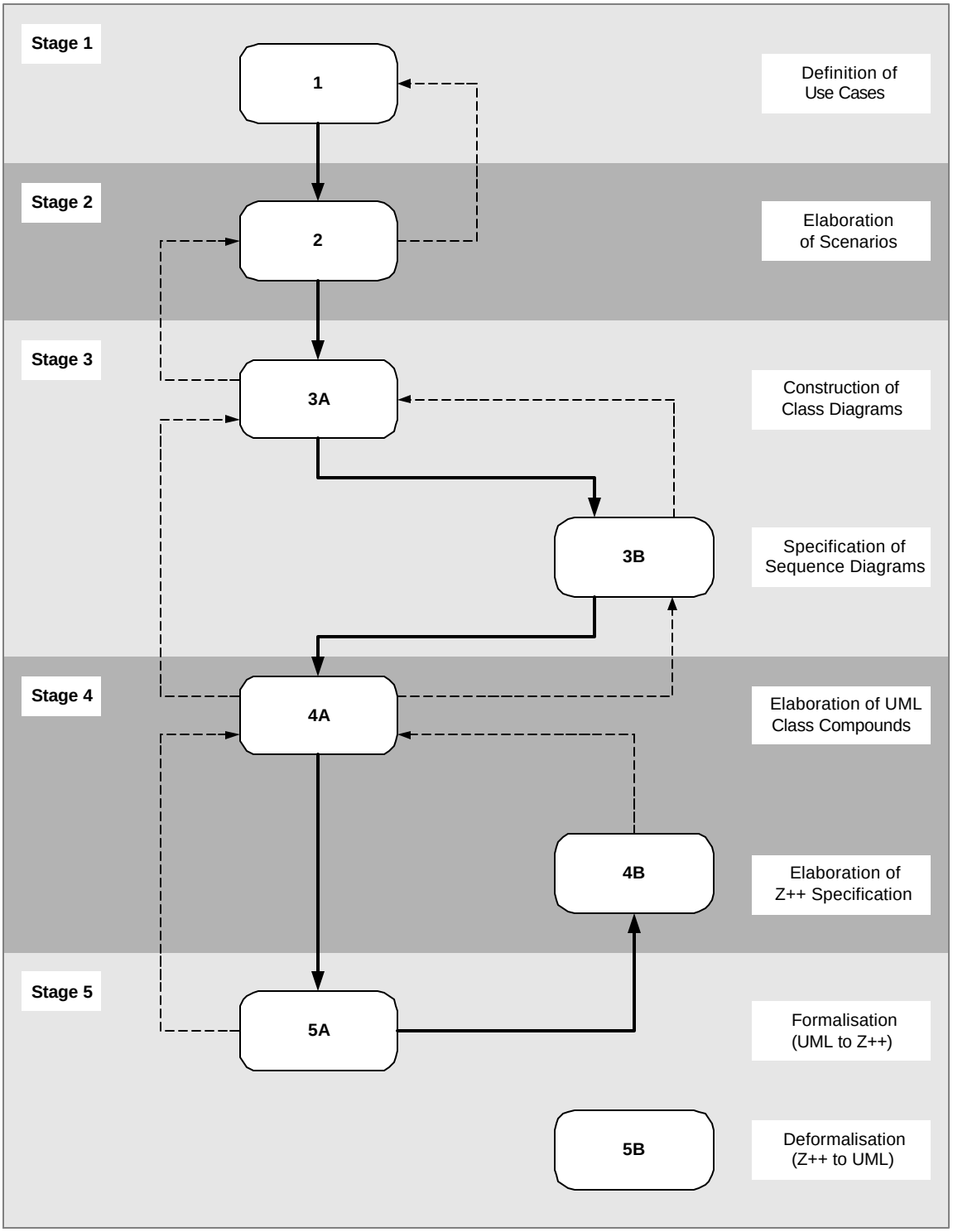

Fig. 7.2 Regular Sequence of Modelling Activities

can be assimilated with the normal scenario of the use case represented by the procedural frame described in Fig. 7.1) The modelling stages are highlighted, the direct flow of activities is emphasised by a thicker, continuous line and the iterative revisions of specifications are indicated by a dashed line. This scenario, which in its "forward segment" (that is, not including feedback links) does not encompass the deformalisation activity (reserved for "irregular" modelling scenarios), can be succinctly described by the <1, 2, 3A, 3B, 4A, 5A, 4B> sequence, where the numbers are associated with activities as indicated in Fig. 7.1.

7.4.5 Alternative Flows of Modelling Activities

The procedural frame presented in Fig. 7.1 encompasses different orderings of activities and we do not claim that the "regular" flow suggested in the previous Subsection represents the unique or the most effective way of developing the integrated UML/Z++ model of the system. There are other alternatives possible, and depending on the particular application, on the experience of the development team, as well as on an a series of other factors, including project priorities and deadlines, one of them may be considered better suited for the particular development needs of a given application. Our "regular" chaining of modelling activities represents only a reference procedure which we believe can be applied in the general case, but nevertheless we do not constrain the ordering of the steps in the modelling process, more important being the correct completion of the integrated model.

Among the other alternatives of sequencing the modelling activities, the one presented in Fig. 7.3 is described here because it highlights a specific strategy that deserves further examination. More precisely, this example of "irregular" scenario for the modelling process can be described in its "forward segment" as <1, 2, 3A, 4A||4B, 5A||5B, 3B>, where the symbol || describes parallel activities (notice that in order to show that 3B comes after 5A and 5B a compromise regarding the notation has been made in Fig. 7.3, where thick dashed lines are used as part of the "forward segment"; they are however different from the regular feedback connections, which continue to be represented as thin dashed lines). Two elements are special in this scenario: first, the fact that the description of classes proceeds in parallel in UML and Z++ and, second, that step 3B comes last in the forward part of the scenario.

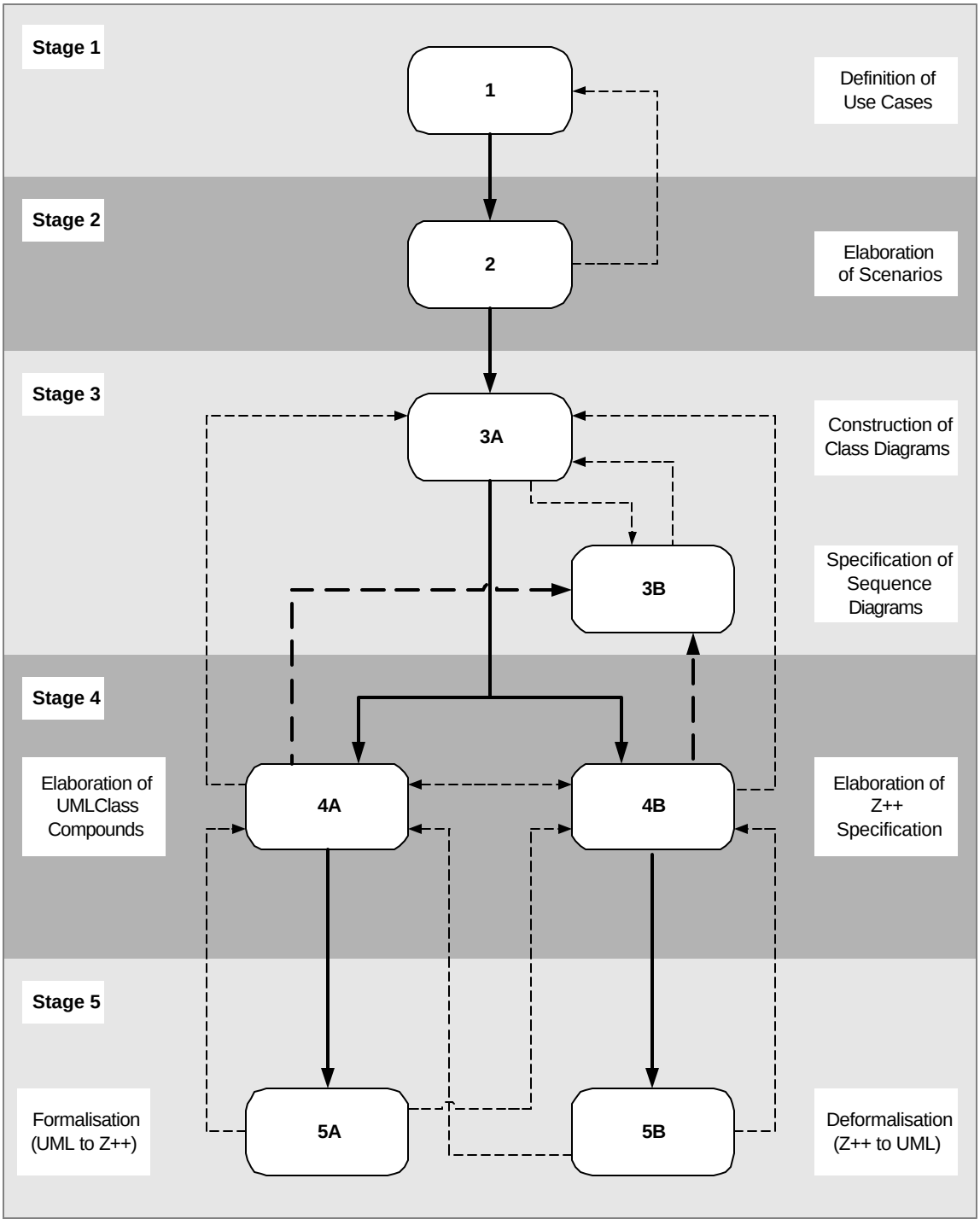

Fig. 7.3 An Example of “Irregular” Flow of Modelling Activities

The first element highlights the idea that various teams of specifiers may have various backgrounds and while some would favour the use of UML, some may prefer employing Z++ as specification notation. In fact, there is the possibility that the specification of classes may proceed first in Z++ and then in UML (and, by extrapolation, it is theoretically possible to have all classes specified in Z++ and not at all in UML). The second element illustrates the idea previously mentioned in Subsection 7.4.3 that sequence diagrams can be used as a tool for fine-tuning the specification, and thus can be the last set of artefacts developed in the modelling process. Of course, additional refinements for improving the accuracy of the model follow in any case.

Another example of an irregular modelling scenario, which stresses rapid development is, in its "forward segment," <2, 3A, 4A, 5A, 4B>, meaning that the definition of use cases (step 1) and the specification of sequence diagrams (step 3B) are omitted. In short, this modelling alternative takes a "fast-track route" and, after the elaboration of scenarios, class diagrams are developed, UML classes are detailed, the formalisation process takes place, and the detailed specification of Z++ classes is completed. It represents in fact a shorter version of the regular flow of modelling activities suggested earlier. Interestingly, perhaps due to space limitation, in many papers describing the use of UML only the artefacts of steps 2, 3A and 4A are described, in some cases step 2 being skipped as well. While we recommend the regular alternative described in 7.4.3 the above modelling scenario can nevertheless work well in various application contexts.

7.5 Chapter Summary

In this chapter a procedural frame for pragmatic, efficient and reliable modelling of TCS have been presented. The artefacts produced in the modelling process, specifically various sets of UML diagrams organised in collections, the set of UML class compounds, and the formal specification consisting of a set of Z++ classes have been presented. The class compound, a simple yet useful construct serving primarily the purpose of formalising UML

classes in Z++ and representing a practical extension of the fundamental notion of UML class, has been introduced. Specific abbreviations that are later used in the development of the Harmony environment have been associated with the modelling artefacts. The modelling activities, included in the proposed procedural frame and organised in five stages have also been described and the modelling process has been discussed in terms of both regular and irregular chainings of activities. The application of the suggested modelling process, relying on the combined use of UML and Z++, is illustrated on an Elevator System case study presented in the next chapter. It has also driven the design of the supporting tool Harmony described in Chapter 9.