4 Related Work

"But search the land of living men
Where wilt thou find their like agen?''

[Walter Scott, Introduction to
Canto First, Marmion, 1808]

4.1 Introduction

The purpose of this chapter is to narrow the research space, focus on the topic location and discuss current specification approaches that are related to ours. Based on the examination of these approaches, the contour of our work can be drawn with greater accuracy, leaving to the remaining of the thesis the task of completing the detailed picture of our approach. Some general observations regarding the integration of notations in software specification are presented first, followed by a brief review of a number of semi-formal/formal combinations of notations involving formalisms other than Z. Then, the examination of integrations of notations is narrowed down to research projects that involve Z or variants of Z. In particular, five approaches that share significant characteristics with the modelling solution presented in this thesis are discussed in more details and both commonalities and differences are highlighted. Because the approach proposed in this dissertation places special emphasis on capturing temporal properties of systems, a review of existing modalities of dealing with time in the context of Z-based specifications is also included.

4.2 Integration of Semi-formal and Formal Notations in Software Specification

Integrating formal with semi-formal or informal notations in software development is not a new idea, some forms of combinations being present in a fair number of approaches. After all, formal languages like Z include provisions for textual, plain language annotations, intended to alleviate the difficulty of following complex mathematical expressions and to relate abstract descriptions with real-world entities. However, as pointed out by several authors, one of the main reasons that, in addition to lack of tools support, have prevented the wider application of formal methods is that not sufficient attention has been paid to the integration of formal techniques with traditional, semi-formal methods [Gerhart94, Clarke96, Lawrence96].

Many authors consider the integration of formal techniques with conventional, informal (or semi-formal) approaches as highly beneficial in software development. For instance, [Aujla94] points out that formal techniques are portable and extendable and can be used in various ways and in various phases of the development. They can be applied as complementary techniques or as alternatives to conventional approaches. Their application leads to the detection of a significant number of errors in specifications. On the other hand, Aujla et al. show that formal techniques themselves benefit from being included in the larger frame of an integrated methodology; they are provided with both context and method, which they may lack if considered in isolation. Alexander sees the combination of formality and informality as a way to obtain “the best of both worlds” [Alexander95] and Bruel et al. point out that “the main objectives of integrated formal/informal approaches is to make formal methods easier to apply and to make informal methods more rigorous” [Bruel98b, pp. 52].

Integration, which in general covers combination of notations, models, and even methods [Bruel98b], has nevertheless its own issues, most notably the fact that interpretations underlying the translation rules from informal to formal are seldom explicitly stated, the

focus of formalisation is in general on basic constructs, and not structures, and little attention is paid to relating the results of analysing the generated formal models to the corresponding components of the informal counterparts (Bernhard Rumpe, in the [Bruel98b] panel).

However, in general, using complementary, concerted techniques for modelling software systems brings a series of benefits, the most important being the increased modelling power provided by the combination and the higher level of confidence they bring in regarding the correctness of the software product being developed. Evidently, these advantages did not pass unnoticed by the researchers and practitioners of the software engineering field, and various combination strategies have been proposed. Some of these strategies are briefly reviewed in the rest of this chapter but, before that, it is useful to point out that, in broad terms, the relationship between the formal and the semi-formal (or informal) components of a specification can be one of the following (notice that we refer in particular to semi-formal or informal graphical notations):

- If the graphical (semi-formal or informal) part is built initially and then a translation process is applied to obtain its formal counterpart, we can speak about derivation of the formal model from the informal model or simply of formalisation (e.g., [Lee95], where diagrammatic and text elements of Bailin's object-oriented requirements specification method OOS [Bailin89] are translated into Z counterparts, or [Laleau00], where the translation is from UML to B). Certainly, it is also possible to obtain a visualisation of the formal part, in which case the derivation is from formal to visual (e.g., [Salek94], where the REVIEW system is used in the larger frame of the METAVIEW meta-system –which facilitates the development of CASE environments– to generate natural language descriptions from Environment Definition Language (EDL)/Environment Constraint Language (ECL) specifications, or [Kim99b], where graphical representations for Z constructs are proposed). The later form of derivation can also be called deformalisation;
- If in addition to diagrammatic representations some related formal specifications are produced independently (e.g., [Jia97], where Z specifications supplement UML models),

the approach can be characterised as complementary formalisation. Typically, this approach also involves derivation from informal to formal, a subset of the diagrammatic description of the system being translated into formal specifications (this is the case for the cited [Jia97] approach, which is discussed in more detail in Subsection 4.5.1);

- if changes in any of the specification's parts are continuously propagated in the other, we can speak of a tight integration of notations (e.g., [RoZeLink99], where UML models are connected to corresponding ZEST descriptions).

In the above classification the terms semi-formal part and formal part of a specification are used but we should point out that, due to the costs involved, formalising the entire specification of a software product is generally impractical, if not impossible, and the typical approach is to apply formal techniques only to the critical sections of the software being developed [Gerhart94]. As such, the correspondence between the diagrammatic (semi-formal or informal) and textual (formal) parts of a specification is typically limited to a subset of the specification's components.

Depending on the number of notations involved, a combination of notations can take the form of either a dual-notation integration (e.g., [Björkander00], where UML is combined with SDL) or of a multiple-notation integration (e.g., the multi-paradigm specification technique devised by Zave and Jackson [Zave96], the pure formal method integration (PFMI) strategy suggested by Paige to allow the combined usage of formal methods such as Z, refinement calculus, predicative programming, and Larch [Paige98], or the framework solution proposed by Day and Joyce for integrating multiple notations [Day00]). Generally speaking, the integration does not necessarily involve a formal/semi-formal (or informal) combination; it can be of the formal/formal type (e.g., [Sowmya98], where the dynamic aspects of RTS are modelled using both Statecharts and FNLOG, a logic-based language built on first-order predicate calculus and TL) or semi-formal/semi-formal (e.g., [Scogings01], where an integration UML/Lean Cuisine+ is proposed for supporting the early stages of interactive system design). And, as mentioned in the classification proposed above, it has also been considered useful to deformalise the formal models [Salek94, Kim99b].

For the purpose of comparing various integration approaches we also introduce the notion of monolithic environment, which means that a single CASE tool is used for developing both semi-formal and formal models, and various subsequent formal processing (such as analysis and refinement) can be invoked from this tool. The alternative, the non-monolithic environment, refers to a combined use of CASE tools, with separate invocations from the operating system.

We believe that the integration of notations does provide a viable solution for modelling complex systems because various aspects of the systems need various ways of description, which can beneficially complement each other (for a classification and examination of forms of method complementarity, primarily in terms of notations and processes, we suggest [Paige99]). In particular, in the case of formal/semi-formal integrations, it is always possible to "fine tune" the formality level and adjust the balance between the less rigorous diagrammatic representations and the formal specifications to best answer the needs of a given application. An important point we must not forget about integration is, as well stated by Clarke et al., that the end result should be a compound, and not a mixture (in other words, a solution in which the components are tightly united, and not one in which the components simply intermingle) [Clarke96].

Within the research space introduced in Section 2.1, we look next at a number of integration approaches that propose a formal/semi-formal combination of notations.

4.3 Semi-formal/Formal Integrations of Notations Not Involving Z

By resorting again to the classification proposed in Table 2.I and to the research space presented in Fig. 21 we can detail now the areas that neighbour our thesis' C3+ topic location and have a look at research approaches that "reside" in these areas. Figure 4.1 contains an enlarged depiction of a major portion of Fig. 2.1, which covers all 16 combinations of integration as defined in Section 2.2 (recall that the classes –or areas– C1•

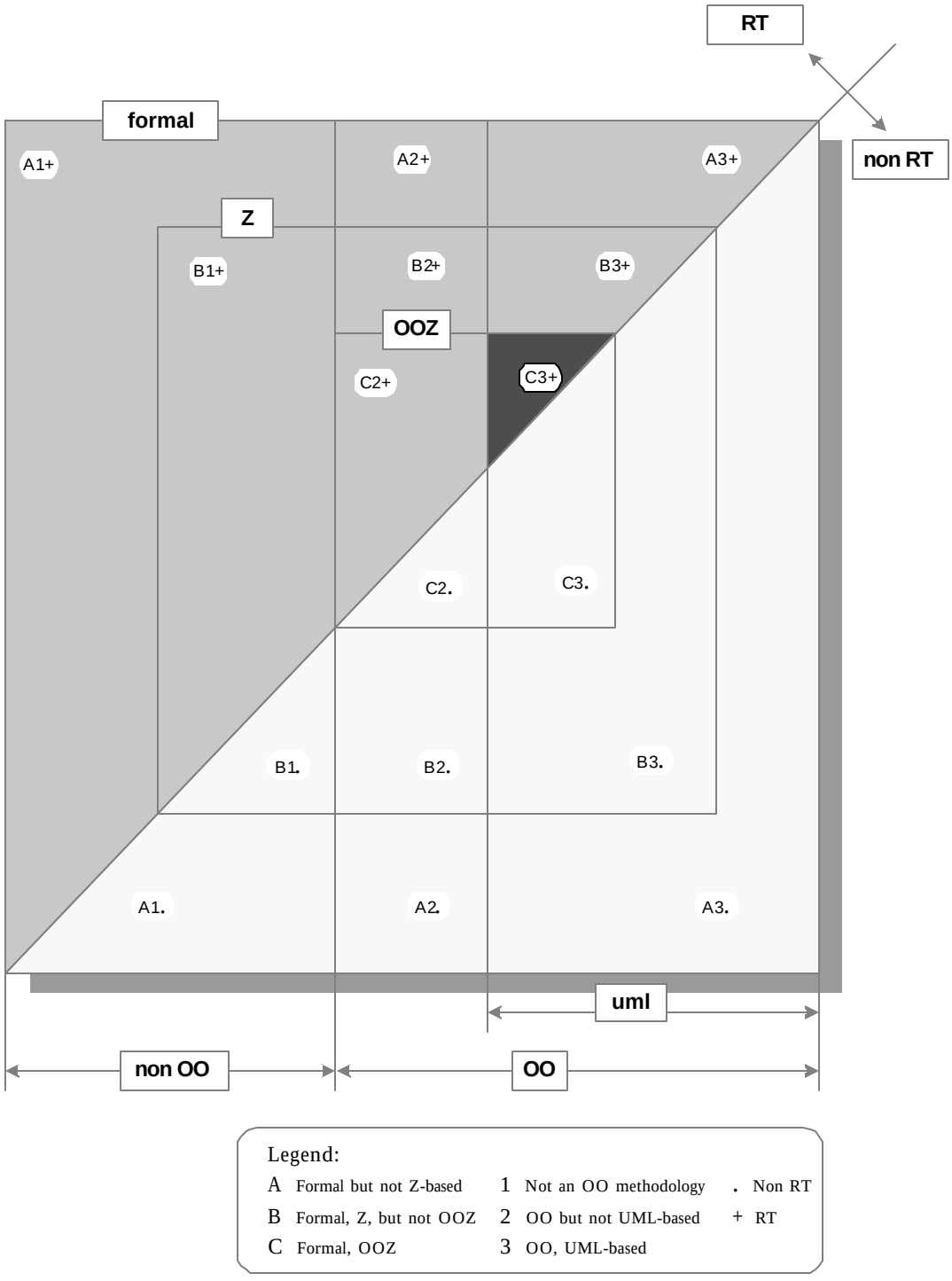

Fig 4.1 First Zoom-In on the Research Space

and C1+ are discarded). In this section we focus only very briefly on some examples of projects that fit in the A1• to A3+ areas, as an introduction to the next section, where Z-based approaches are discussed. This introduction is intended to be simply illustrative and by no means comprehensive (areas A1 to A3 are quite large, because they include "everything but not Z" of all possible semi-formal/formal combinations of notations that cover our research space). Examples of A-type approaches, with succinct descriptions, are presented in Table 4.I. It can be inferred from this table that the topic of semi-formal/formal integration has been pursued constantly by researchers, and no remote area ("remote" in the sense defined by our classification) has been left uncovered.

Table 4.I Examples of Semi-formal/Formal Integrations Not Involving Z

Area	Area Characteristics	Example Approach	Summary Description of the Example
A1•	Formal non-Z, Non OO, Non RT	[D'Almeida92]	Translation from Modified Entity-Relationship diagrams (MER) and textual Keyboard-based Formatted Descriptions (KDF) to VDM
A1+	Formal non-Z, Non OO, RT	[Sahraoui97]	DFD-based methods integrated with TL constructs of the Zaman language [Sahraoui92]
A2•	Formal non-Z, OO non UML, Non RT	[Cheng94]	The VISUALSPECS environment supports the formalisation of OMT models in algebraic languages such as Larch
A2+	Formal non-Z, OO non UML, RT	[Chen98]	Integration of HRT-HOOD (Hard Real Time- Hierarchical OO Design) models [Burns95] with TAM (Temporal Agent Model) specifications [Scholefield92]
A3•	Formal non-Z, UML, Non RT	[Laleau00]	B specifications generated from UML diagrams
A3+	Formal non-Z, UML, RT	[Bordbar00]	Petri Nets serve for representing and analysing dynamic models in a UML-based approach aimed at modelling discrete-event dynamic systems

4.4 Semi-formal/Formal Integrations of Notations Involving Variants of Z

After the introduction to semi-formal/formal integrations of notations based on examples that do not involve Z, it is now time to look at the closer neighborhood of the thesis topic, represented by areas B2• to C3+, which make up the "Z sub-domain". The best way to do this is to enlarge again the original representation of Fig. 2.1 and discard the peripheral areas A1• to A3+, thus resulting the depiction shown in Fig. 4.2. Examples that serve the

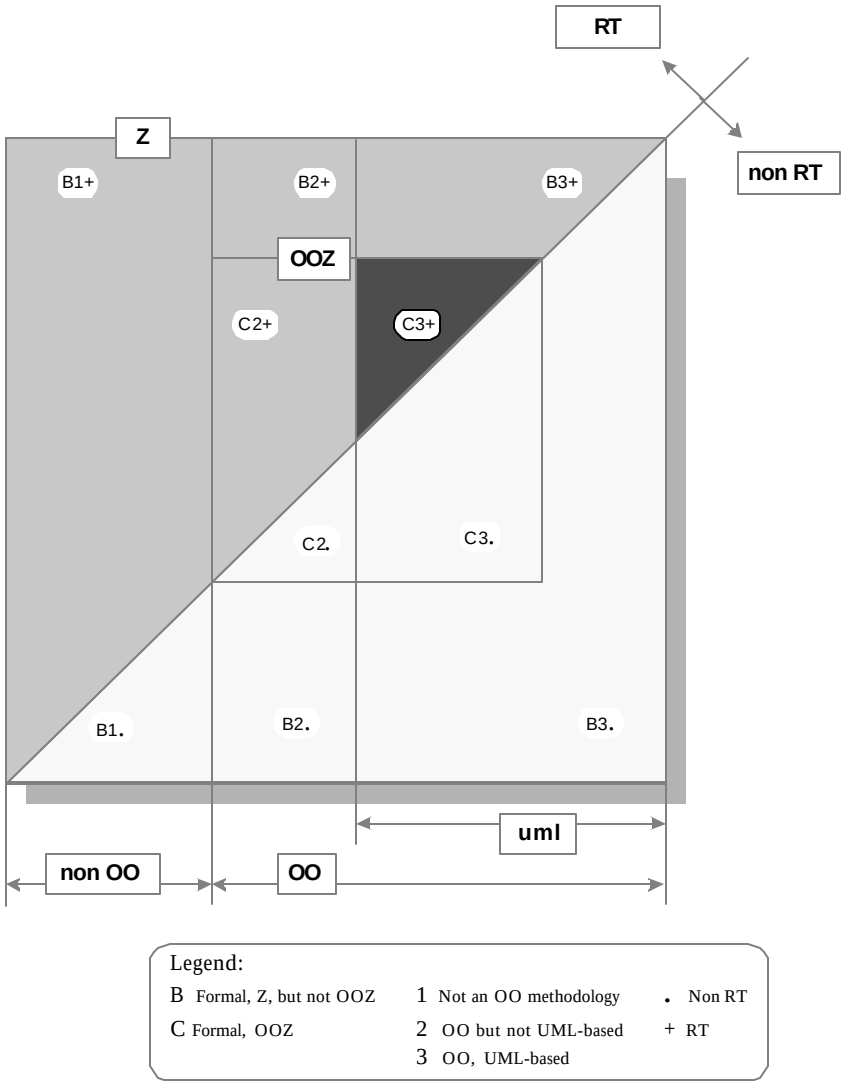

Fig. 4.2 Second Zoom-In on The Research Space

illustration of integration classes proposed in Chapter 2 are given again in tabular form (Table 2.II). Regarding the completion of this table, it can be noted that even though all ten areas of the "Z sub-domain" have been covered, examples for some classes have been more difficult to find than for other. In particular, the example for B1+ is the only one we find after a rather long search (typically, in the earlier approaches, when Z was integrated with notations of structured approaches, the focus was not on RT applications). Also, for the C2+ category we had to resort again to [Lano95], the only other candidate we found being [Dong97b], but there the addition of an OMT description to the Object-Z specification of a multiple-elevator controller is rather accidental, and not suggested as an integration approach per-se. The closer categories B3 to C3 are also not very populated, and in fact the few approaches that fit in these areas of the thesis' topic's "near vicinity" constitute the more restricted group of "closely related approaches," discussed next at the last and most detailed level of investigation of the thesis' research space.

4.5 Closely Related Approaches

While, as previously shown, there are numerous approaches that integrate in various degrees graphical, semi-formal representations with formal notations, very few are aimed at explicitly dealing with TCS using a Z-based formalism incorporated in the larger frame of the OO paradigm. We have identified five specific approaches that in our view are the closest to the direction of work that we have pursued. However, of the five approaches, only two include provisions for explicitly dealing with temporal properties of the systems, as we also have attempted.

4.5.1 Jia's Augmented Object-Oriented Modeling Language

Xiaoping Jia, the author of the well-known Z type checker ZTC [Jia98a], takes a pragmatic approach in combining the strengths of semi-formal graphical OO notations with those of formal specifications [Jia97, Jia98c]. The author indicates that only a partial automation of

Table 4.II Examples of Semi-formal/Formal Integrations Involving Z

Area	Area Characteristics	Example Approach	Summary Description of the Example
B1•	Z but not OOZ, Non OO, Non RT	[Aujla94]	ERD and DFD formalised using Z within the Rigorous Review Technique (RRT)
B1+	Z but not OOZ, Non OO, RT	[Coombes92]	Formalisation in Z of casual timing diagrams (diagrams inspired from those used by electrical engineers to illustrate temporal properties of digital devices)
B2•	Z but not OOZ, OO but not UML, Non RT	[Lee95]	Constructs of Bailin's OOS method transformed into equivalent Z specifications (also mentioned in Section 4.2)
B2+	Z but not OOZ, OO but not UML, RT	[Bruel96]	Fusion models translated into Z specifications (precursor of [France97] shown in the B3+ area)
B3•	Z but not OOZ, UML, Non RT	[Jia97] [Noe00]	Formalisation in Z of UML constructs (details in Subsection 4.5.1 and, respectively, 4.5.2)
B3+	Z but not OOZ, UML, RT	[France97]	Structural and behavioural Octopus analysis models expressed in UML are formalised using Z (details in Subsection 4.5.3)
C2•	OOZ, OO but not UML, Non RT	[Nguyen96]	Proposal of a 4-submodel specification based on the integration of OMT and Object-Z* (a slightly modified version of Object-Z); RT properties not explicitly targeted
C2+	OOZ, OO but not UML, RT	[Lano95]	Formalisation of OMT constructs in Z++ (more details in Chapter 6)
C3•	OOZ, UML, Non RT	[RoZeLink99]	Two-way link between UML constructs supported by Rational Rose 98 and ZEST specifications (details in Subsection 4.5.4)
C3+	OOZ, UML, RT	[Kim00b]	UML and Object-Z combine forces for describing a lights control system (details in Subsection 4.5.5)

code generation can be achieved from semi-formal OO models, in the form of a skeletal implementation. Thus, in his approach formal notations are used for partial description of the system, as a complement of the traditional OOAD models (as indicated in the definition of complementary formalisation proposed in Section 4.2). The approach is driven by practical reasons and its aim is to minimise changes and extensions of widely-used semi-formal and formal notations while providing an intuitive and easy to use, yet powerful software development method. Specifically, Jia proposes a language denoted AML (Augmented Object-Oriented Modeling Language) that essentially combines notations from UML and Z. For pragmatic reasons, minimal additions to the Z notations have been included (mostly for handling the specification of classes), making up the slightly extended Z notation referred to as Zext. A supporting tool called Venus was developed to provide the very useful capabilities of model analysis, animation of a large subset of Z specifications, refinement of the design based on a fixed, yet comprehensive library of data structures and algorithms, and extensive C++ code generation.

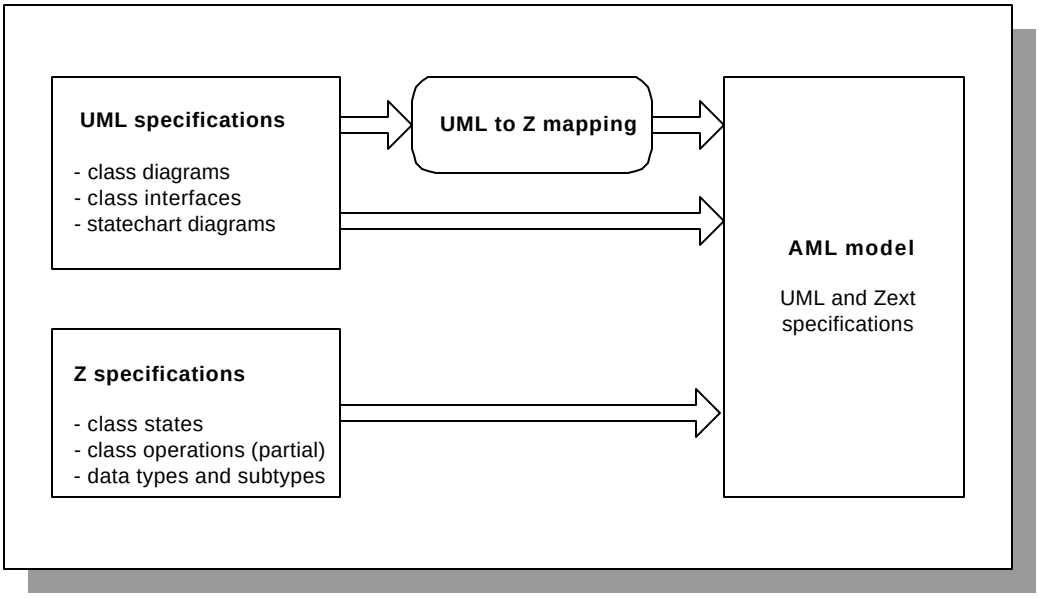

Fig. 4.3 Jia's AML-based Approach

Jia's AML-based approach can be related to the relatively new research direction of light-weight formal methods, succinctly discussed in Subsection 2.6.5. As described by Jia, the

integration of UML and Z is focused on compensating for the limitations of UML, primarily the fact that data types and operations are not formally specified. In essence, as shown in Fig. 4.3, adapted from [Jia97], the UML notation is used to specify the system's class organisation, the class interfaces, and the related state diagrams, while the Z language is employed to provide supplementary details, specifically class states, data definitions, and partial descriptions of operations.

While excellently addressing the practical barriers that hinder the large-scale use of formal methods in practice, Jia's approach differs from ours in a number of ways. Firstly, there is no particular emphasis on capturing time-related property of systems, thus making its application dependent on the modeling ability of UML and on the limited time-capturing capability of the regular Z. Secondly, even though the UML notation is employed, the formal part of the object-oriented model is expressed via a minimal set of extensions of Z, and we believe that by employing a full-fledged object-oriented version of Z additional modeling power would be available, without significant increase of the notation's complexity. Thirdly, it is not indicated whether the opposite translation, the mapping from Z to UML is included. The diagram on which we have based Fig. 43 indicates that Z specifications are only fed forward to the complete model, without a corresponding feedback from the integrated AML model to Z descriptions.

4.5.2 Noe and Hartrum's Extension of Rational Rose 98

More recently, Capt. Penelope Noe, from the Air Force Personnel Center, Randolph, Texas, and Prof. Thomas Hartrum, from the Air Force Institute of Technology (AFIT), Ohio, have proposed the extension of Rational Rose 98 for the inclusion of formal specifications [Noe00]. In summary, their approach is to exploit existing features of Rational Rose, specifically Rose's scripting language and available textual fields that can be used for embedding formal expressions, and produce a formalised model that can be fed into the AFITtool transformation system. Based on this input, the AFITtool is capable of generating Ada code (Fig.4.4). From Rose's set of graphical representations, only the class diagrams and the state diagrams are considered, and an additional non-Z and non-UML state transition

table is also employed in the semi-formal specification process. Using primarily the Documentation field associated with classes, operations, and state transitions, Z specifications that supplement the description of the system can be embedded into the extended Rose model. These formal specifications are partially written in the $L^{A}T_{E}X$ format.

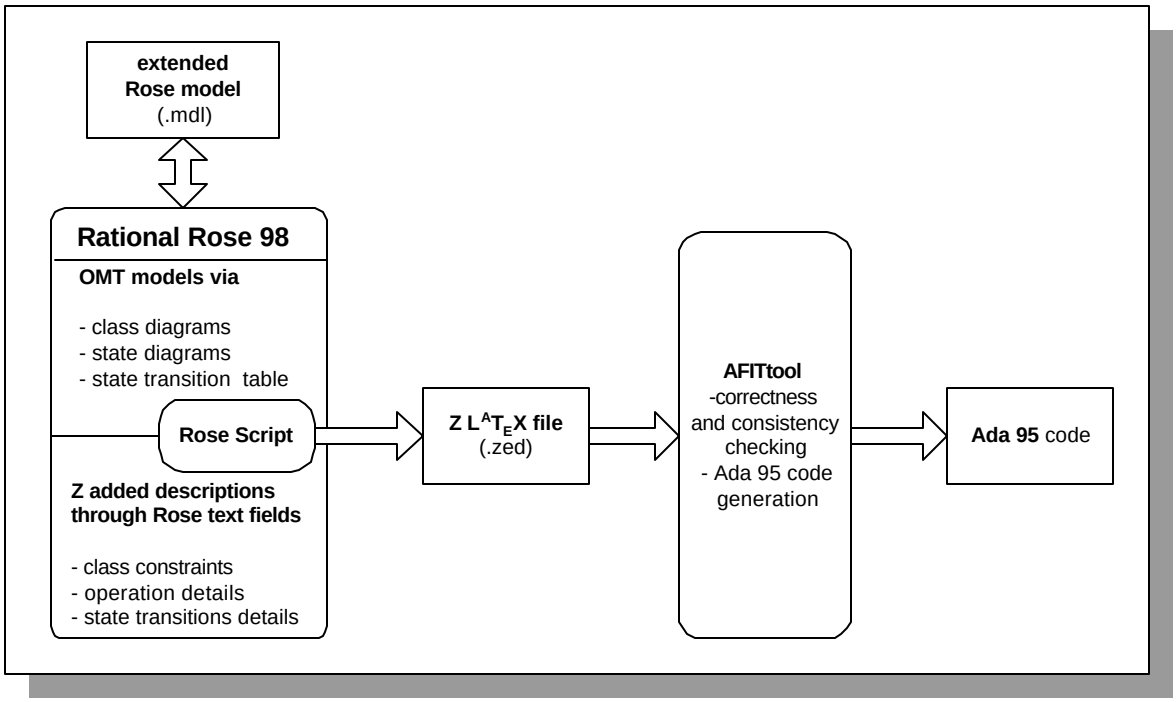

Fig. 4.4 Noe and Hartrum's Approach

The approach follows the OMT methodology [Rumbaugh91] and consists of building three models: the object model, which defines the class structure of the application, the functional model, which describes the desired interaction of the system with its environment, and the dynamic model, which expresses the state changes of the system. In summary, the object model is created by augmenting the UML class diagram with Z-specified user defined data types and constraints on classes and attributes, the dynamic model is built using regular finite state machines whose states and events are represented in Z using static schemas, and the functional model is obtained through the description of class operations, details such as pre– and post–conditions being added in Z and the operations being represented by dynamic Z schemas. After the extended model of the system is completed a translation procedure (a

Rose script) is invoked in order to produce a Z file in $LaTeX$ format, as entry for the AFITtool. Consistency and correctness checks are performed and Ada 95 code is produced.

This approach is well explained in the [Noe00] paper and its practical utility has obvious merits. In addition, as indicated by the authors, it suggests a viable line of work, that of developing Rose scripts for interfacing with other CASE tools. This approach is different from ours in several ways. Firstly, as in the [Jia97] approach discussed previously, RTS are not targeted explicitly. Secondly, Z is used again in its regular version, which has the advantage of keeping the notation simple and the potential of interfacing with a larger variety of analysis tools, but this solution is less direct than employing an OO variant of Z for OO specifications. Thirdly, many Z descriptions are entered in the $LaTeX$ syntax, which is clearly not user-friendly. Fourthly, the internal format of the ".zed" file is custom-made (tailored to the AFITtool), and thus its usage in connection with other tools is restricted. In addition, as in the [Jia97] example, this approach also fits in the complementary formalisation category of integration, and ours proposes a tight-integration solution. Lastly, it can be noted that although Rose 98 acts as the sole front-end modelling tool, Noe and Hartrum's integration of notations is not entirely monolithic. This is due to the fact that a separate program, the AFITtool, with its own set of commands and interface demands, is invoked outside the main environment, Rational Rose.

4.5.3 Blending Octopus and Z

The approach described by France et al. in [France 97] represents one of the relatively few attempts of integrating OOAD methods with formal specification techniques for developing RTS (work connected to this approach is described as well in [Bruel96], [Shroff97], and [Bruel98a]). The formal specification language used is Z, which was chosen, as indicated by the authors, because of its maturity and the availability of related analysis tools. Here, the combining of an OO approach with a formal specification technique consists of translating the three analysis models of the Octopus method [Awad96] into equivalent Z specifications. The formal specification language Z is used to enhance the modelling capability of Octopus

by allowing consistency checking across models (thus opening the way for the application of automated analysis tools) and by providing the developer with a better insight into the problem's requirements. Octopus analysis models are translated into Z constructs using procedures that could be partially automated. The formalisation process is applied to all three analysis models of Octopus: the object model is formalised using class schemata, while the derivation of the other two models, dynamic and functional, which capture system behaviour, involves four steps: definition of states, definition of subsystem responses to events, Z modelling of transitions described in the dynamic model, and description in Z of statechart actions and activities, including those represented in the functional model (Fig. 4.5, based also on the earlier [Bruel96] paper on FuZE, which combines Fusion and Z).

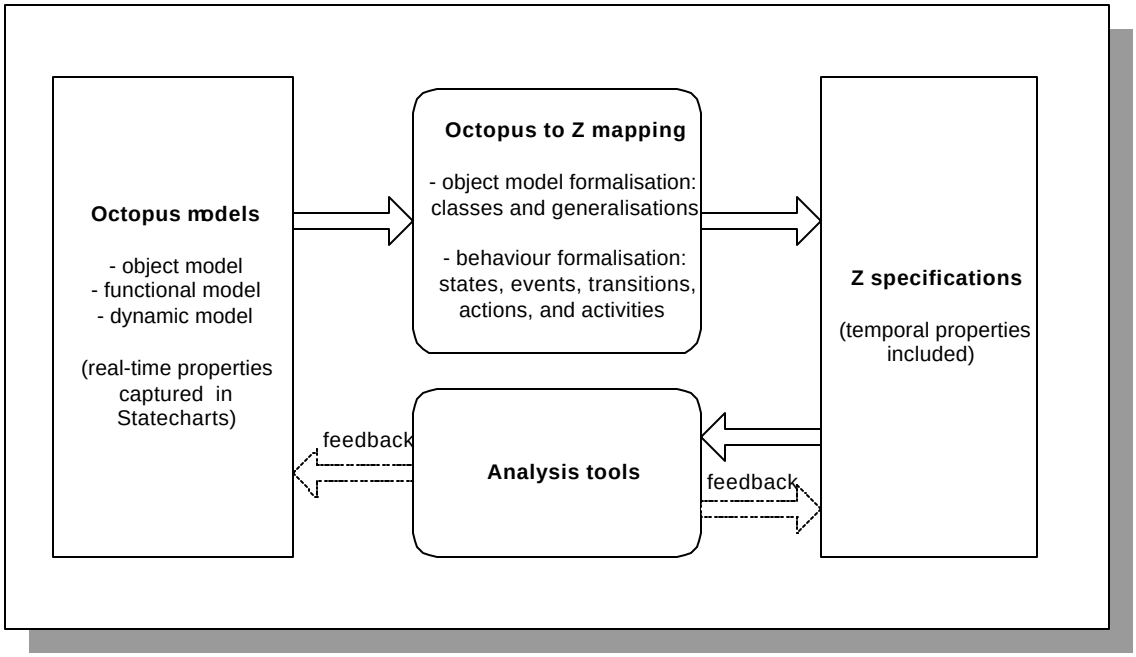

Fig 4.5 The Octopus and Z Integration Approach

Feedback from analysis tools to both the diagrammatic and the formal models is considered, but we can note however that in this approach the integration of notation is not tight in the sense defined in Section 4.2. Also, even though the object model is translated into regular Z constructs that model classes, a translation into an object-oriented version of Z would be more natural and direct, and the specifications would have similar structure in terms of

classes. Additionally, the OMT-based notation of Octopus has currently less exposure than UML, which has enjoyed a constantly growing expansion over the last few years.

4.5.4 Headway System's RoZeLink

Most probably, the only tool that has been developed commercially to support an object-oriented modeling approach and combine the advantages of graphical, semi-formal notations with those of formal notations is RoZeLink [RoZeLink99], produced by Headway Software Inc. as a bridge between the UML notation supported by the 1998 version of the Rational Software Corporation's Rose environment and the ZEST object-oriented formal

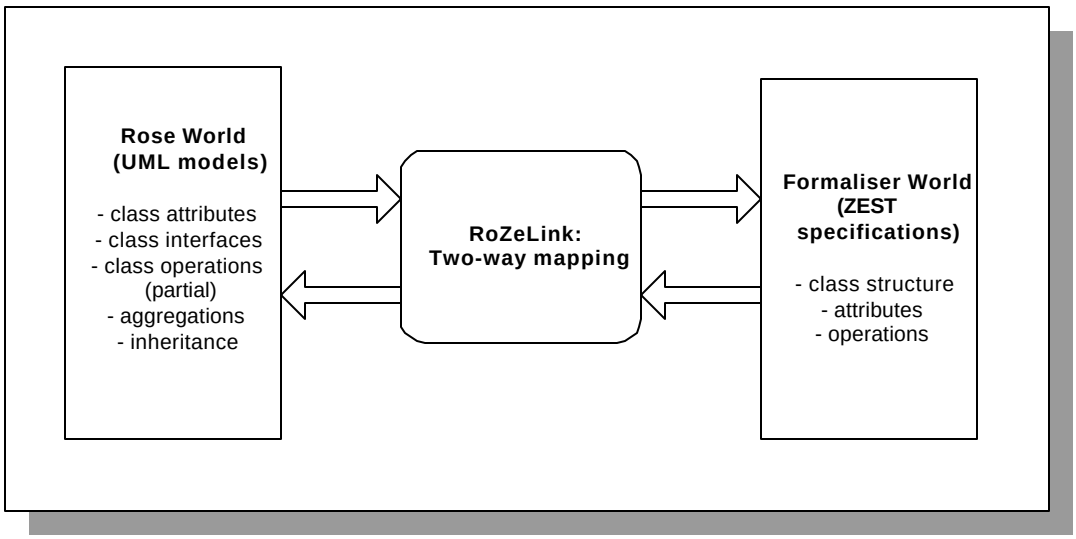

Fig. 4.6 The RoZeLink Tool

specification language supported by Logica's Formaliser [Formaliser01]. RoZeLink, which apparently has not been further developed since the producing company has changed its direction of work (see the web-site in the [RoZeLink99] reference), provides the necessary bi-directional link between the two notations and achieves the goal of maintaining specifications consistent between models. RoZeLink operates between a Rose UML model and a collection of Formaliser documents, and through a continuous translation mechanism maps elements from the semi-formal model into the formal model and vice-versa. This

implies that changes in one "world" are reflected during the modelling process in the other one (Fig. 4.6). We borrowed from RoZeLink this idea and also pursued a tight integration of notations.

Although practical and comprehensive in its dealing with structural aspects of the system, the Headway Systems' approach has its limitations, primarily because there are no particular provisions for dealing with time-related properties of the systems. In addition, only the class structure of the system is involved in formalisation, the statecharts are not. Also, the RoZeLink approach does not propose a truly monolithic integrated environment, its role being to act as an intermediary that interconnects two already existing commercial software development tools. In order to work both "graphically" and "formally" on his specifications, a user must first start up three separate applications.

4.5.5 Object Z and UML

The closest approach to ours (it belongs in the same C3+ class) and also the most recent is presented in [Kim00b] (earlier work by the same authors on formalising UML diagrams is described in [Kim99a] and [Kim00a]). In many ways, our work is similar to Kim and Carrington's alternative for integrating UML and an object-oriented variant of Z, but there are also some notable dfferences, as indicated below. First, however, we would like to indicate that we started to develop our approach in the form presented here sometime in 1998 and an early outline of the integration and of the proposed Harmony tool was presented by the author of this thesis in August 1999 as part of the requirements for the Visual Languages course taught by Prof. Phil Cox at Dalhousie University, Halifax, Nova Scotia [Dascalu99]. Therefore, we have worked independently in the same topic area, and only very recently have learned about Kim and Carrington's approach.

In summary, as shown in Fig. 4.7, their proposal is to translate UML models into Object-Z models, temporal properties of the systems receiving adequate treatment via a time trace notation based on time refinement calculus. Kim and Carrington's approach proposes not

only a formalisation of the class structure but also a formalisation of dynamic properties based on use case diagrams, sequence diagrams, and statechart diagrams. Very briefly, there is a direct correspondence from UML classes to Object-Z classes that "makes the semantic translation between the two languages less complex" [Kim00b, pp. 241], and the dynamic behaviour of the system is formalised using detailed translation rules for all elements of the statecharts (initial state, regular states, entry and exit actions and activities, events, and guards).

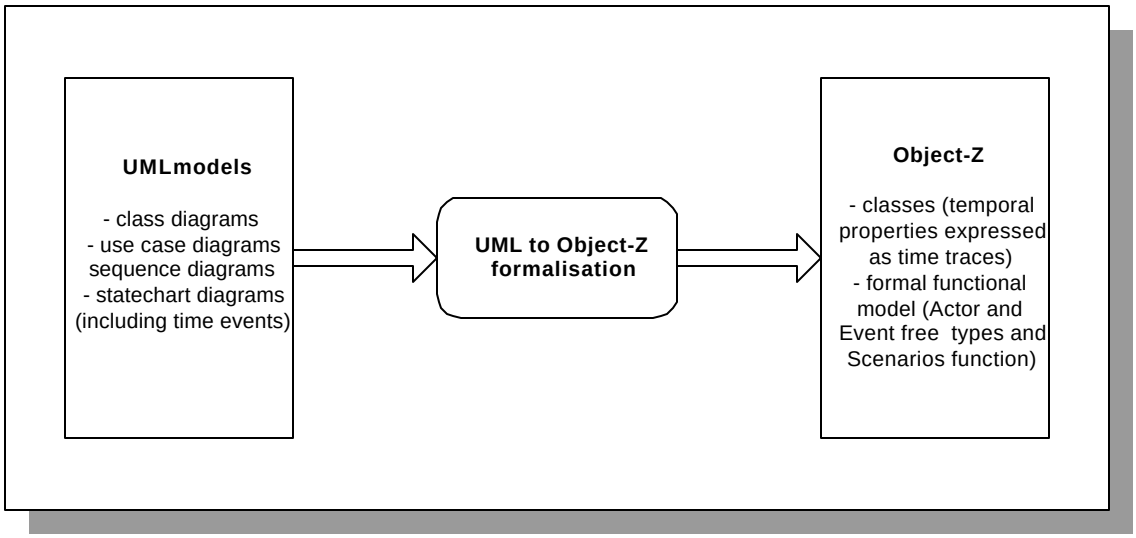

Fig. 4.7 The UML/Object-Z Combination

Kim and Carrington's approach is one of the most mature solutions for UML and Z integration and has been developed in one of the strongest research groups on formal methods, the Software Verification Research Center at Queensland University, Brisbane, Australia (their web-site is mentioned in the [Cogito97] reference). It builds on extensive research developed over more than a decade by prominent scientists in the field, and benefits from a suite of tools and techniques that have been validated through numerous applications. Nevertheless, our approach also has its merits. It uses Z++ instead of Object-Z and we believe that Lano's OOZ proposal fares better than Object-Z in some respects, specifically in the details included in the class specification (see class definition in Subsection 3.2.3) and in the integration of the RTL semantics and syntax for explicitly dealing with temporal

properties. In our view, RTL, described in Chapter 5 in conjunction with our formalisation process, has an intuitive and natural syntax and its semantics are easier to grasp by developers not trained in formal methods. Z++ and RTL offer therefore a friendlier user-interface and sustain our lightweight alternative for pragmatic TCS specification. Also, the approach presented by Kim and Carrington does not propose a tight integration of notations (there is not a two-way mapping between semi-formal and formal models) and there is no specific mention of a development tool in their paper, so we cannot ascertain its characteristics under the monolithic/non-monolithic environment criterion.

4.6 Modalities of Specifying Temporal Constraints in Z

Capturing time-related properties of systems is not a simple task. Actually, it is one of most demanding challenges faced by the developers of timed-constrained systems, as emphasised by the profusion of approaches proposed in this direction, including numerous variants of Temporal Logic (comprehensively reviewed in [Bellini00]) and all sorts of "timed" formalisms, including Timed Petri Nets [Ramchadani74], Timed CSP [Schneider92], Timed CCS [Moller92], Timed Statecharts [Kersten92], and Timed LOTOS [Léonard98], (Interestingly, although various alternatives of using Z for specifying RTS have been proposed, the term "timed Z" appears nevertheless in surprisingly few references –and in a rather general way,– so we cannot speak of an established Timed Z notation.) Since a general discussion of the various solutions proposed over the years for capturing temporal properties of systems exceeds the scope of this dissertation we summarise in this Section only some of the most important ways of tackling the "time issue" within Z-based specification approaches. In general, enhancement of Z with constructs and symbols borrowed from other formalisms have been proposed (e.g., [Mahony92, Fidge97, Mahony98, Yuan98]), much fewer being the approaches that attempt to capture timing constraints using exclusively the constructs of regular Z (e.g., [Evans97]). In all cases, special mechanisms for modelling temporal properties have proved to be necessary.

4.6.1 Time Refinement in Z

In one of the earlier approaches that employ Z for modelling time-constrained systems, Mahony and Hayes propose an extension of the notation and the use of refinement calculus to allow a unified treatment of both analog and discrete properties of such systems [Mahony92]. Three "notational devices" are introduced: topologically continuous functions for expressing both analog and discrete quantities, physical units attached to variables, and specification statements describing the assumptions made by the system about its environment and the effect the system is expected to achieve provided the assumptions are satisfied. In order to declare a variable such as the temperature at a given location (say, in an aquarium), a `TEMPERATURE` type with associated physical units can be first defined by resorting to the set $\mathbb{R}$ of real numbers:

$$\texttt{TEMPERATURE} == \mathbb{R}\ \texttt{[Celsius]} \tag{4.1}$$

Then, by introducing the type `TIME` through syntactic equivalence with the set of real numbers (and with an appropriate physical unit attached):

$$\texttt{TIME} == \mathbb{R}\ \texttt{[Second]} \tag{4.2}$$

the evolution of our variable of interest can be modelled as a continuous total function:

$$\texttt{aquariumTemperature} : \texttt{TIME} \rightarrow\!\!\!\!\ominus\ \texttt{TEMPERATURE} \tag{4.3}$$

On the other hand, discrete variables, such as the following one, which indicates whether or not fresh water is pumped into the aquarium, can be modelled as a partial continuous function from time to Boolean ("partial" because they are possibly times when the variable is undefined or its value is changing):

$$\texttt{waterIn} : \texttt{TIME} \rightarrow\!\!\!\!\oplus\ \mathbb{B} \tag{4.4}$$

Other important concepts used in Mahony and Hayes' approach include the notion of open time interval (α ... β), the collection τ_{TIME} of all open intervals of time, the time topology T_{TIME}, which encompasses all periods of time consisting of open, disjoint intervals, the `cov(Period)` function, which gives the collection of disjoint, open intervals of time that comprise the `Period` set of time, and timed history predicates such as `Pred on Period`, `Pred in Period`, and `Pred at t`, where `Pred` is a logical predicate, `Period` a period of time consisting of a set of open intervals of time, and `t` a moment in the passage of time.

This approach allows the description of both analog and discrete properties of systems –two aspects of TCS that usually are modelled separately– within a unified framework that at the same time supports the capturing of temporal properties over intervals of time. By associating physical units to variables it helps both the understanding and the type compatibility checking of the specifications. While this approach is expressive and practical for specifying various properties of RTS, including concurrency aspects, it has been pointed out that by modelling variables as constant functions over all time the focus is shifted away from important features of Z, such as state schemas and operation schemas, which become "buried in the specification" [Dong97a, pp.26].

4.6.2 The Quartz Alternative

The Quartz approach [Fidge97] is similar to the previously described work of Mahony and Hayes in that it deals with time and functional behaviour in a unified way, and places equal emphasis on capturing temporal constraints and on specifying functional requirements. The proposed scope of the Quartz approach is however different since it aims at integrating RTS specifications (in a variant of Z) with program refining techniques leading to the generation of Ada-like high-level programs augmented with time constraints. In the words of its authors, "Quartz encompasses real-time software development from specifying the formal requirements through writing the high-level language code" [Fidge97, pp. 100]. The major principles of the method are that program development and verification are performed in lockstep at all levels of abstraction and the same rules are applied throughout the entire

refinement process. In outline, Quartz proceeds as follows: first, top-level specifications are created, defining the behaviours of the system via allowable traces of observable variables (histories indexed by the absolute time); then, the specifications are refined to a set of concurrent components that constitute the basis for a skeletal program design; next, each individual component is refined using sequential refinement rules, leading to the identification of low-level state changes and descriptions in an executable subset of the specification notation that correspond to constructs of the target high-level language; finally, since some time constraints may not be yet fully verified, they are subject to further analysis at the executable code level.

Conceptually, time is modelled in Quartz using an additional variable that in the refinement process receives the same treatment as the other variables of the system do. The time domain can be either discrete ($T == \mathbb{N}$) or continuous ($T == \mathbb{R}$) and the variable `now` can be introduced to model the passage of available processor time. The concepts of action systems are brought in to allow the expressing of concurrency and timing and, from the notational point of view, Z schemata are combined with guarded-command language constructs.

The goal of the authors of Quartz, that of proposing a formal development method that iteratively transforms top level RTS specifications into executable time-verified executable code is undoubtedly ambitious, but what strikes the reader of the [Fidge97] article is the complexity of the approach, a relatively simple example necessitating a rather long refinement and long explanatory descriptions. Of course, this is the general case with formal refinement and analysis, but questions can be raised regarding the applicability of the method in all but smaller-sized or highly critical applications.

4.6.3 Andy Evans' Approach

Andy Evans also shows that even though traditionally it has been considered that Z in itself is insufficient for specifying RTS, it is nevertheless possible to introduce extensions to the standard Z language that provide the capability of capturing the dynamic aspects of the

systems [Evans97]. In his approach, Evans proposes four extensions to Z addressing the issue of specifying reactive systems: genericity, generic operations being used as instruments for describing concurrent behaviour; real-time extension, allowing the specification of dynamic properties of the system, including timed computations; modularity, that permits the encapsulation of concurrent behaviours; and synchronized communication, which allows modules to communicate via a CSP-like mechanism. Notably, Evans uses only regular Z constructs, thus eliminating the need for specialised specification and analysis tools.

The key idea of Evans' approach is to specify the dynamic behaviour of the system as the set of allowable sequences of system states. States and operations are specified in the classical Z style, thus providing the static specification of the systems, while the dynamic specification is achieved using a model based on the notions of infinite computations, atomic events, and non-deterministic interleaving of atomic operations. Infinite sequences are specified using a new data type (in the following, X is a type):

$$\mathtt{comp}\ X == \mathbb{N}_1 \rightarrow X \tag{4.5}$$

a next-state schema is introduced, and a generic operation validcomp is proposed as an extension to Z for specifying the valid behaviours (computations) of the system:

```
┌─── [STATE] ──────────────────────────────────────────────────────┐
│_validcomp_ : comp STATE ↔ (ℙ STATE x (STATE ↔ STATE))           │
├──────────────────────────                                        │
│∀σ : comp STATE; I : ℙ STATE; R : STATE ↔ STATE •                 │
│        σ validcomp (I, R) ⇔ σ(1) ∈ I  ∧                         │
│          (∀n : ℕ₁ • σ(n) R σ(n+1) ∨ σ(n+1)= σ(n))               │
└──────────────────────────────────────────────────────────────────┘
```
(4.6)

Timed computations explicitly capturing temporal constraints imposed on the system are modeled using the notions of discrete time (Time == ℕ) and of infinite sequences of states with associated time values. A generic relation $\mathtt{validcomp}_t$, similar to the one in (4.6), is proposed in order to specify the allowable behaviours of the system. Evans' approach makes an elegant use of generic constructs to provide Z with extensions for specifying real-time

systems. However, using infinite sequences of states to describe the system's dynamic properties brings a level of mathematical complexity that may hinder the adoption of the proposed extensions by the larger community of software developers.

4.6.4 RTOZ

Another approach that employs a variant of Z, specifically Object-Z, in a formalism aimed at specifying RTS is Periyasamy and Alagar's Real-Time Object Z (RTOZ) [Periyasamy97, Periyasamy98]. RTOZ addresses both time-dependent data and time-constrained processes, and allows for a separation between temporal constraints and functional specification. The philosophy of RTOZ is based on the notion of filter specifications (classes that specify timing constraints) and a model of time that relies on the history of data objects.

A specification in RTOZ is composed of two sets of classes: regular classes, that capture the structural and the behavioural requirements of the system without regard to temporal restrictions, and timing classes (filters) that model timing properties of the system. There is a one-to-one correspondence between regular classes and filter specifications, a filter specification describing the timing constraints imposed on the behaviour of its associated class. Each filter specification consists of several filter schemas, and each operation in a given class is restricted by a filter schema in its class' associated filter specification.

The approach described by Periyasamy and Alagar is novel in that it utilizes real-time filters in the context of object-oriented specifications, and makes a clear separation between the specification of the system's functional requirements and the description of its timing constraints. This demarcation between the timing aspects and the "time-abstracted" behaviour of the system brings a series of advantages, most notably the increased reusability of the functional specifications of the system, localisation of effects in the case changes in requirements are required (improved by the isolation of functional requirements from temporal constraints), and better understanding of both the operational characteristics and

the timing properties of systems. Also, RTOZ extends only minimally the syntax of Object-Z, thus preserving the capabilities of Object-Z without increasing its complexity.

Although RTOZ provides adequate support for the verification of properties such as safety and liveness, it can nevertheless be difficult to specify the characteristics of TCS only in a formalised way. A combination of diagrammatic and formal techniques would combine the advantages of both, essentially ease of use on the one hand and rigorous, verifiable descriptions on the other.

4.6.5 TCOZ

Another approach aimed at capturing RT requirements is presented in [Mahony98] and [Mahony00], and involves the combination of Object-Z and Timed CSP in a blended notation called Timed-Communicating Object-Z (TCOZ). The motivation of this notation is, as pointed out by its authors, to complement the expressive modelling power of Z regarding the static, single-threaded specification of systems with the capability of Timed CSP of capturing the behaviour of concurrent real-time systems. The integration of notations and techniques is actually multi-levelled; first, Object-Z extends Z with constructs suitable for object-oriented modelling, then the notion of time is added to Object-Z to obtain the enhanced notation Timed Object-Z. This enhancement is made possible by considering a global real-time clock, represented by the state attribute `now`, and by modelling environmental interactions as functions of time, included in the system state. On the other hand, CSP is extended with two primitives, `delay` and `timeout` that permit the specification of temporal aspects of sequencing and synchronisation. Finally, Timed Object-Z and Timed CSP are blended in the TCOZ notation, whose principal characteristic is to model operations as terminating CSP processes and objects as non-terminating processes.

The basic constructs of Timed CSP are sequencing, parallel composition of processes, and choice (internal and external). Sequencing has two forms, the first one describing the succession event-process behaviour as follows:

$$a \ @ \ t \rightarrow P(t) \tag{4.7}$$

where a is an event, t is the time parameter , and P the process.

The second one describes the sequential composition of processes, as in:

$$P;Q \tag{4.8}$$

where the sequential execution of processes P and Q is indicated by the operator "**;**".

Parallel composition of processes is represented using the syntax:

$$P \ |[X]| \ Q \tag{4.9}$$

where P and Q are processes and X is a set of events enabled jointly by P and Q.

The external choice operator has the form:

$$a \rightarrow P \quad b \rightarrow Q \tag{4.10}$$

and signifies that the above processes begins by enabling both a and b and then behaves (as P or Q) according to the event a or b that is actually enabled by the environment. The internal choice operator has a similar meaning, but the variation in behaviour is determined by the internal state of the process:

$$a \rightarrow P \sqcap b \rightarrow Q \tag{4.11}$$

The above operators are added to Z's set of operators and bring with them the semantics of CSP. The time-specific primitives `delay` and `timeout` are as well imported in the extended version of Object-Z.

In essence, the approach proposed by Mahony et al. makes use of the complementary semantics of the state-based behavioural model and of the event-based behavioural model and offers an excellent example of multi-integration of formal notations for software specification. However, the very combination of the two extended formalisms may raise a barrier that could prevent the wider acceptance of TCOZ in practice; the result is a rather complex notation, not easily accessible to developers who are not trained in formal methods. Also, oversized specifications may result from applying TCOZ to larger systems.

4.6.6 Other Approaches

Besides the approaches discussed above, other proposals for applying Z to TCS have been made over the years. Some of them are succinctly reviewed below.

In one of the earliest approaches, Duke and Smith suggest the integration of Z and TL for modelling TCS in a solution that allows the verification of properties such as liveness and safety [Duke89] but as indicated in [Johnson95] the application of temporal operators on both schemas and predicates can be confusing.

The work of Coombes and McDermid [Coombes93] can also be placed in the traditional line of research, that of enhancing the semantics of Z with semantics of other formalisms that are more suitable for specifying and verifying TCS. In essence, the authors consider constructs specific to a variant of TL, namely Interval Logic, employ the grid concept to allow the inclusion of multiple clocks (needed in distributed systems), and adapt to time intervals the CSP concept of trace. Although sound and thorough, Coombes and McDermid's approach seems too complex for practical application and can lead to oversized specifications.

The issue of capturing temporal properties using Z is also addressed by C.W. Johnson, this time in a less researched context, albeit very important, that of supporting user interface development in the construction of interactive safety-critical systems [Johnson95]. Johnson's proposal combines Z schemas with TL formulae, structured graphics, and generic input events. While Johnson's proposal successfully addresses a series of issues pertaining to the formal development of user interfaces (such as modelling of temporal properties that affect usability and synchronisation between the interface and the underlying application) and is supported by a prototyping system entitled Prelog there is still work needed regarding the refinement of specifications, as acknowledged by the author.

Dong and Zucconi suggest a framework for incorporating time in Z-based formal models [Dong97a] and propose the use of timed refinement and the ProCoS approach [He96] to capture the input environment and the Quartz approach to express the requirements of the core system, all within the frame of an extended version of Object-Z. This is one of the most flexible frameworks proposed to date for extending the modelling power of Z to the RT domain, since it allows the integration of a variety of time formalisms (not only the ones mentioned above) in an OO extension of Z. Nevertheless, the observations made previously regarding TCOZ can apply here as well.

In a similar line of research, involving the expression of time constraints in a Z-centered formalism, Bolognesi and Derrick introduce an ambitious concept, constraint- and object-oriented (C-O-O), in a highly innovative specification method that in essence combines object-oriented constructs (mapped to Object-Z) with constraints that define the time-ordering of operations (modelled as transition graphs) [Bolognesi98]. In our opinion this solution, although very original and interesting, is too complex and involves an adjustment of the OO paradigm that may appear too difficult to the larger community of software developers.

Also relatively recently the proposal of a Complete-Object-Oriented-Z (COOZ) has been made [Yuan98], relying on an object-oriented version of Z that integrates mechanisms and notations from Object-Z and OOZE and employing Duration Calculus (DC) [Chaochen91] for describing temporal properties of objects. Yuan et al.'s solution is one of the most complete proposals to date and its application is supported by a set of tools, entitled COOZ-Tools, that consists of an editor and viewer, a syntax and semantics checker, a refinement tool, a help system, and a project manager. Although DC is considered by the authors of COOZ more powerful than TL, it is the very complexity of DC and the particularities of its notation that can constitute an obstacle for the larger application of COOZ in practice.

4.6.7 The Z++ Alternative

As mentioned in Subsection 2.7.3, Z++ supports the modelling of TCS by incorporating a TL-based formalism. In essence, the HISTORY clause of the class specification describes the admissible sequences of execution, in the form of TL or RTL predicates. Because we rely on Z++ to achieve "time capturing," the Z++ way for dealing with time is described in more detail in Chapters 5 and 6 of the thesis. We mention here only that our time specification solution relies on Jahanian and Mok's RTL, whose constructs are incorporated in the larger frame of Z++ in the way proposed initially by Lano [Lano95]. This solution follows the general approach for extending Z to TCS modelling, that of incorporating constructs and symbols from other formalisms, and has been chosen for reasons outlined in Chapter 5 of the thesis.

4.7 Chapter Summary

In this chapter work related to our approach has been surveyed. The major directions of integrating notations in software specification have been investigated and a closer look at proposals aimed at dealing with systems characterised by complex temporal properties has

been taken. The major ways of dealing with time in software specification have been identified and several particular approaches have been analysed in greater detail. As the overall result of our survey, we found out that five reported projects come significantly close to the line of research we have pursued; they are, respectively, Jia's pragmatic approach based on AML, Noe and Hartrum's support for formal methods in Rational Rose, France et al.'s formalisation of Octopus, Headway Software's RoZeLink tool, and Kim and Carrington's integration of UML and Object-Z. The major characteristics of these approaches have been discussed and the main differences between them and our own approach have been pointed out.