HighSpec: a Tool for Building and Checking OZTA Models

J.S. Dong P. Hao X. Zhang
National University of Singapore
{dongjs,haoping,zhangxi5}@comp.nus.edu.sg

S.C. Qin
University of Durham, UK
shengchao.qin@durham.ac.uk

ABSTRACT

HighSpec is an interactive system for composing and checking OZTA specifications. The integrated high level specification language, OZTA, is a combination of Object-Z (OZ) and Timed Automata (TA). Building on the strength of Object-Z's in specifying data structures and Timed Automata's in modelling dynamic and real-time behaviors, OZTA is well suited for presenting complete and coherent requirement models for complex real-time systems. **HighSpec** supports editing, type-checking as well as projecting OZTA models into TA models and Alloy Models so that TA model checkers-UPPAAL and the Alloy Analyzer can be utilized for verification. Most importantly, **HighSpec** supports a novel yet effective mechanism advocated by OZTA for structural TA design, i.e., using a set of composable timed patterns to capture high level timing requirements and process behaviors and generate the TA part of model in a top-down way. **HighSpec** can also generate LaTeX document as an alternative media for the spread and read of established OZTA models.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications —*Languages, Methodologies, Tools*

General Terms

Languages, Verification

Keywords

Object-Z, Timed Automata, Structural Design, Verification

1. INTRODUCTION

The specification of complex real-time systems requires powerful mechanism for modelling data structure, concurrency and real-time behavior as well as tool-support for building up and verifying the established models.

HighSpec is an interactive system for composing and checking OZTA models. OZTA [3], is an integrated formal modelling technique for specifying real-time complex systems, which combines Object-Z [8] and Timed Automata [1]. Building on the strength of Object-Z's in specifying data structures and Timed Automata's in modelling dynamic and real-time behaviors, the OZTA specification language is well suited for presenting complete and coherent requirement models for complex real-time systems. Besides the combined modelling power bought from Object-Z and traditional Timed Automata, OZTA supports structural TA design in a top-down way by introducing a set of timed patterns as language constructs to specify complex real-time behaviors. This set of TA patterns is defined in our previous work [4] based on TCOZ [7] constructs such as *deadline*, *timeout*, *waituntil*. They were initially formulated as an interchange media to achieve a translation of TCOZ models to TA models so that TCOZ models can be model-checked by reusing TA's powerful tool supports (e.g., UPPAAL [6]), but then found also useful for TA domain alone in supporting systematical TA designs. The semantics of these patterns was later incorporated into OZTA semantics and hence provides a solid foundation for tool development. Detailed information on the patterns can be referred in our previous work [5].

HighSpec supports all the modelling features of OZTA and meanwhile provides powerful checking capabilities. The main functionalities are listed as following,

- Automated systematic TA design via timed pattern,
- Schema editing and expansion,
- Syntax and type checking,
- Projection to TA model checker, UPPAAL, for verification,
- Projection to Alloy for analysis,
- Generation of LaTeXpresentation of established models.

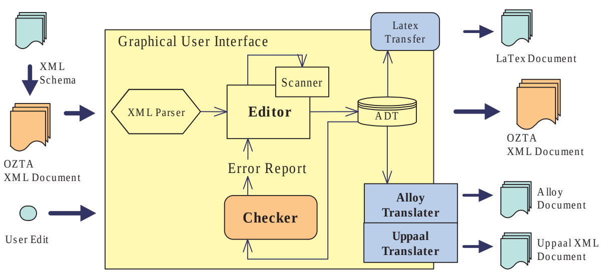


Figure 1: Overview of HighSpec

Figure 1 provides an overview of **HighSpec**: It mainly consists of five components, i.e., a powerful GUI editor to compose Object-Z schemas and the corresponding timed automaton, a syntax and type checker, a LaTeX code generator for the spread and read of established OZTA models and model translators to UPPAAL and Alloy for verification.

2. MODELLING

HighSpec provides powerful automated support with user commands for directing the design of an OZTA model. All the information input from the user interface are collected into a special Abstracted Data Type (ADT) designed according to the integrated syntax of Object-Z and Timed Automata. Basically, these information can be divided into three parts. The *system configuration part* supports declaration of global information and classes. Each OZTA class contains an *Object-Z part* and a *Timed Automaton part*. The *Object-Z part* contains the information of Object-Z schema such as state variables, operations and pre/post condition of operations, and the *Timed Automaton part* captures the information about the control flow between the Object-Z operations and related timing behaviors according to system requirements.

2.1 Structural TA Design

The Object-Z schema information recorded in ADT plays an important role for designing the corresponding timed automaton. Once the definitions of Object-Z operation schemas in an OZTA class are completed, its corresponding timed automaton can be generated in a top-down way by repeatedly applying the timed patterns to fulfil the control and timing requirements. **HighSpec** suggests the following guidelines for the structural TA design using the timed patterns.

2.2 Guidelines for TA Design Using Patterns

Many complex control and real-time behaviors can be naturally modelled as collections of small processes lying in different layers of the systems, operating and interacting sequentially or concurrently. In OZTA specification, if a process is small enough and its internal behavior can be abstracted away at the desired specification level, it then can be seen as an atomic process. Usually, these atomic processes are firstly defined in an OZTA class as Object-Z operations. Then a Timed Automation is used to capture the control flow and timing requirements involved with these operations, each of these Object-Z operations in an OZTA class has a mapping in its corresponded TA part, which is an atomic state.

The generic TA patterns, in a way, provide a set of templates to decompose a complex real-time behavior into different layers and smaller processes. By repeatedly applying the timed patterns to fulfil the control and timing requirements, we can eventually achieve the concrete timed automaton in a systematic way.

There are certain guidelines for the engineers to use these generic timed patterns for structural TA design in **HighSpec**:

- Decide the layers of a complex control and real-time behavior. An abstracted automaton will be substituted by its inside layer by applying appropriate patterns. The TA model of a system can finally be generated in a top-down way by continuously embody its inside layers using appropriate patterns.
- Decompose the complex processes of one layer into smaller processes with simple behaviors. those smaller processes mostly are composed together by sequential composition or alternation.

2.3 Generating TA Using Timed Patterns

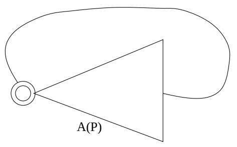
In this subsection, we will use a timed queue example to demonstrate how the timed patterns can be applied to facilitate TA designs in a systematic way.

The essential behaviors of this timed message queue system is that it can receive a new message through an input channel '*in*' within a time duration 'T_j' or remove a message and send it through an output channel '*out*' within a time duration 'T_l'. If there is no interaction with environment within a certain time 'T_o', then a message will be removed from the current list.

According to the requirements, it is easy to identify that there should be two operation defined in the Object-Z part of the OZTA model, i.e., *Add* and *Del* and four kinds of patterns should be applied to capture the control and timing behaviors, i.e., *recursion*, *timed out*, *external choice* and *event prefix*.

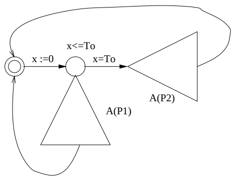
The timed queue automaton can be generated step by step in a top-down way as follows:

step 1. The outmost layer is a recursion of process A(P), which captures a repeated behavior of receiving and removing messages. The triangle represents an abstracted automaton; the left vertex represents the initial state of the automaton; and the right edge represents the terminal state of the automaton shown in figure (a).



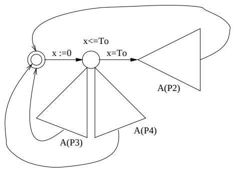
(a)

step 2. The abstracted automaton A(P) are now substituted by a timeout pattern. $A(P1)$ stands for the behavior before timeout and $A(P2)$ stands for the behavior if timeout occurs as shown in figure (b).



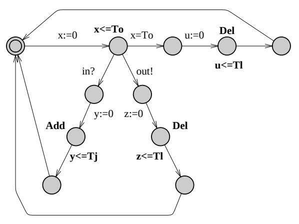
(b)

step 3. The *external choice* pattern is now applied to decompose $A(P1)$ into two smaller components, $A(Pp3)$ and $A(P4)$, which respectively representing the *Add*, and *Del* behaviors with timing constraints.



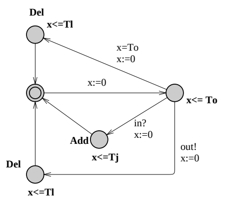
(c)

step 4. Processes $A(P3)$ and $A(P4)$ both match the timed event prefix pattern and the deadline pattern. The atomic operations Add and Del are modelled as TA states. Clocks x, y, z and u are generated to give these operations the timing constraints. Now the system has been embodied into a concrete timed automaton as shown in figure (d).



(d)

step 5. The last step is to simplify the resultant automaton. Any two consecutive initial and terminal states are merged into one state. In order to reduce the state space of the automaton, the clock y, z and u are replaced by clock x after resetting its value to 0. The final picture of the TA part of the OZTA model for the timed queue system is shown in figure (e).



(e)

HighSpec supports the automation of the above designing process by interacting with the user for determining patterns and the corresponded timing parameters.

3. CHECKING

An OZTA syntax and type checker is implemented in **HighSpec** for checking the validity of an OZTA model as well as model translators for verifying various properties of OZTA models by reusing TA and Alloy's tool support.

3.1 Syntax and Type Checker

The OZTA language has quite a complex syntax as a derivation from Z notation. It is easy, especially for an inexperienced OZTA user, to make some syntax or type errors. **HighSpec** is able to detect and report such errors. A full set of type checking rules can be found in our technical report [2]. The class diagram of this checker is shown in Figure 2.

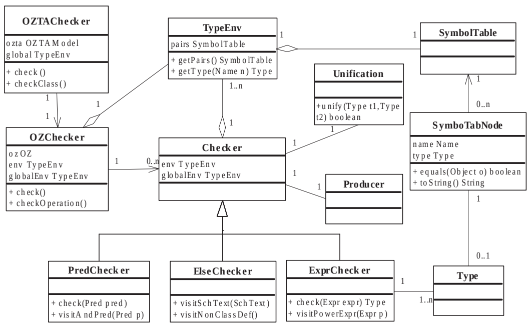


Figure 2: Class Diagram of the type checker

3.2 OZTA to UPPAAL

HighSpec adheres to light-weight principles: instead of implementing a model checker for OZTA models from scratch, we choose to project the integrated requirement models into multiple domains so that existing specialized tools in these corresponding domains can be utilized to perform the checking and analyzing tasks. UPPAAL [6] is a useful integrated tool for verification of real-time systems. By projecting an OZTA model to a TA model, we can reuse UPPAAL to simulate the dynamic behaviors of the OZTA model and verify its various kinds of properties.

The translation process can be automated by employing XML/XSL technology. In our previous work [9], the syntax of Z family languages, i.e., Z/Object-Z/TCOZ, has been defined using XML Schema and supported by the ZML tool. As the UPPAAL tool can read an XML representation of Timed Automata, the automatic projection of the OZTA model (in ZML) to TA model (in UPPAAL XML) is implemented in our OZTA tool.

The uppaal translator in **HighSpec** takes in an OZTA specification represented in XML, and outputs an XML representation of a Timed Automata specification which has its own defined style file DTD by UPPAAL. The automatic transformation is achieved firstly by making use of our OZTA ADT to easily extract information from the specification. A TA interface is then built according to the UPPAAL document structure, e.g., each TA document contains multiple templates and each template contains some states, their transitions and transition conditions. The outcome of our translator is UPPAAL's XML representation of TA, which is ready to be taken into UPPAAM as input for future verification and simulation.

Although our projection can handle most of the TA information of an OZTA model, one limitation needed to be pointed out is that: coupled with operation schema predicates and data structures, the semantics of operation states in the TA part of an OZTA model is slightly different from those of states in UPPAAL. However, the main structure of the OZTA automata model is still consistent with that of UPPAAL model by regarding the OZTA operation states as abstracted automatons which need further implementation. This gap between the OZTA's TA model and UPPAAL's TA model can be remedied by some manual work on the operation states, namely, to further embody these abstracted automatons by adding the data information.

3.3 OZTA to Alloy

Alloy is a structural modelling language based on first-order logic, for expressing complex structural constraints and behavior. The Alloy Analyzer(AA) supports two kinds of automatic analysis: simulation, in which the consistency of an invariant or operation is demonstrated by generating an instance; and checking, in which a consequence of the specification is tested by attempting to generate a counterexample. The essential construct of alloy are signature, fact, function, predicate and assertion.

Projection Rules

Since Alloy is a subset of Z, Our projection is mainly focus on transforming the OZ part of an OZTA model into Alloy. Mapping rules for some important primitives of OZ are presented as follows.

- Types: All the basic types of OZ are defined as, **sig** *TypeRef*{}{*sigFact*} in Alloy.
- State variables: The State variables of an OZ class are projected as fields of an signature **sig***State*{}. However, not all state variables are necessarily to be projected into alloy model, those of which are irrelevant with the properties to be checked in Alloy can be abstracted away.
- The predicates of state schema are projected as facts in Alloy. The keyword **disj** in Alloy can be used to indicate that the variables declared in OZ state schema are disjoint.
- Functions: OZ Functions can be defined as predicate or function in Alloy. When an OZ function returns value, then it should be projected as a function in Alloy, Otherwise it should be projected as a predicate in Alloy. Each parameter of OZ functions corresponds to one parameter in Alloy models.
- Operation schema: Each operation schema *Op* with an empty Δ-list is projected as a predicate; each OZ operation schema *Op* with a Δ-list, $\Delta(s)$, is projected as a fact *Op* in Alloy. The predicates of those operation schemas can be projected as a predicate $pred(s, s')\{\}$. Each state variable in the Δ-list of an OZ operation schema corresponds to a pair of parameters of same type in the Alloy predicate, which respectively represent the pre-state and pose-state of the OZ state variable. i.e.,

  ```
  fact Op{ all s, s': State{ pred(s,s') } }
  ```

- Ordering of state transitions: To reason OZ in Alloy, when the ordering of the state transition of OZ is necessarily to be analyzed, we need import an module *ordering* in Alloy to record the instances of OZ state schemas. The instance of ordering *ord* will record pairs of pre-state and post-state for OZ state variables.

  ```
  open util/ordering[State] as ord module util/ordering[elem]
  one sig Ord {
     first_, last_: elem,
     next_, prev_: elem -> lone elem}
  ```

  The OZ *Init* operation is now projected as a fact which constrains the first element of *ord* in order to record the initial state of the OZ state variables.

4. LaTeX CODE GENERATOR

This generator outputs the LaTeX source file and EPS files as an alternative media for the spread and read of an OZTA model. These source files can be directly complied and viewed in LaTeX tools such as WinEdt.

5. CONCLUSION

We developed **HighSpec**, an interactive tool, for modelling and checking of OZTA, a combination of Object-Z (OZ) and Timed Automata (TA).

HighSpec is a distinguished tool support for modelling and verification of complex real-time systems in that:

- it is built for a high level integrated formal method which provides powerful mechanisms for capturing various aspects of a complex real-time system such as data structure, concurrency and timing constraints. Many of the integrated formal methods such as TCOZ have no exclusive tool support even just for the editing, not to say verification tools. This is mainly because integrated high level specification languages usually contain various aspects of abstracted system information, which makes the implementation of verification tool from scratch impossible. The problem is circumvented in **HighSpec** by projecting the integrated OZTA model to TA and Alloy model so that TA and Alloy's tool support can be reused for checking.
- it implemented a novel mechanism for establishing TA models in a systematical way by using a set of timed patterns to specify common control and timing behaviors. Comparing to traditional TA tool support such as UPPAAL, **HighSpec** effectively release users' trouble to manually cast these common behaviors into clock variables, states, and transition conditions so that they can focus on the specification level of a model rather than the implementation.

Acknowledgements

We would like to thank Chen Qian, and He Kang for their part of work on the coding of **HighSpec**.

6. REFERENCES

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] J. S. Dong, P. Hao, S. C. Qin, and X. Zhang. OZTA. Technical report TRC6/05, School of Computing, National University of Singapore, 2005.

[3] J.S. Dong, R. Duke, and P. Hao. Integrating Object-Z with Timed Automata. In *The 10th IEEE International Conference on Engineering of Complex Computer System*, Shanghai, China, 2005.

[4] J.S. Dong, P. Hao, S.C. Qin, J. Sun, and W. Yi. Timed Patterns: TCOZ to Timed Automata. In *The 6th IEEE International Conference on Formal Engineering Methods*, Seattle, USA, 2004.

[5] J.S. Dong, P. Hao, S.C. Qin, and X. Zhang. The Semantics and Tool Support of OZTA. In *The 7th IEEE International Conference on Formal Engineering Methods*, Manchester, UK, 2005.

[6] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer*.

[7] B. Mahony and J. S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, February 2000.

[8] G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.

[9] J. Sun, J. S. Dong, J. Liu, and H. Wang. A Formal Object Approach to The Design of ZML. *Annals ol Software Engineering*, 13:329–356, 2002.