Verification of Object-Z Specifications by Using Transition Systems: Application to the Radiomobile Network Design Problem

Pablo Gruer, Vincent Hilaire, and Abder Koukam

Université de Technologie de Belfort Montbéliard
4 rue du château
90010 Belfort Cedex, FRANCE
Pablo.Gruer@utbm.fr

Abstract. This paper presents a technique for verifying specifications which uses the object-oriented state-based language Object-Z. The technique is based upon translation of Object-Z specifications into transition systems. The translation of Object-Z into a transition system allows one to use established techniques and tools in order to verify the specifications. We present the basis of our translation approach and then illustrate it by a case study. The case study consists in proving properties of our antennae parameter setting problem specification.

1 Introduction

Formal specification must fulfill two roles. The first is to provide the underlying rationale for the system under development. The second is to guide subsequent design and implementation phases. For example, the model checking approach may be very useful in order to verify a formula related to the specification. Furthermore this verification can be carried out automatically using model checking tools. While most specification formalisms are suited for the first role, few enable automatic verification of systems properties.
Among specification formalisms, Z [7] has received considerable attention. Furthermore, there exist object oriented extensions of this formalism, such as Object-Z [1], which allow for class structured specifications. These formalisms are well suited for concise and comprehensible description of software but automatic analysis tools are very rare, and often restrict to syntactic checking.
Our approach consists in translating an Object specification towards a transition system. Transition systems are behavioral models of software systems which can provide an operational semantic to specifications. Moreover, with transition systems, one can use established techniques [5] and tools, such as STeP [4], in order to prove some properties of Object-Z specifications. Nevertheless, defining an operational semantics for Object-Z is a difficult task. The richness of the specification language, that benefits from the first order predicate notation, is one of the reasons of the difficulty. For instance, operations described by quantified predicates often produce a set of transitions rather than a single transition.

T. Maibaum (Ed.): FASE 2000, LNCS 1783, pp. 222–236, 2000.

Yet, the set of transitions represents an atomic operation and, in order for the semantic to be correct, this atomicity must be preserved in some fashion.
This paper presents a verification technique based on translation rules of Object-Z specifications into transition systems. Rather than an exhaustive presentation of the transformation rules, we address some difficulties that originate in the richness of the specification language. We do not present a formal treatment of this question in this paper, but illustrate it with the case study. Consequently, this work should be considered as the result of an experience on verification of Object-Z specifications by means of transition systems, with STeP as the verification tool. The general organization of this paper is as follows. Section 2 introduces Object-Z and transition systems. Section 3 presents the main idea of the verification technique, i.e., transforming an Object-Z class into a transition system. Section 4 presents the antenna parameter setting problem, its specification and its verification with the STeP environment. Eventually section 5 concludes with an outline of future research.

2 Basic Concepts

2.1 Object-Z

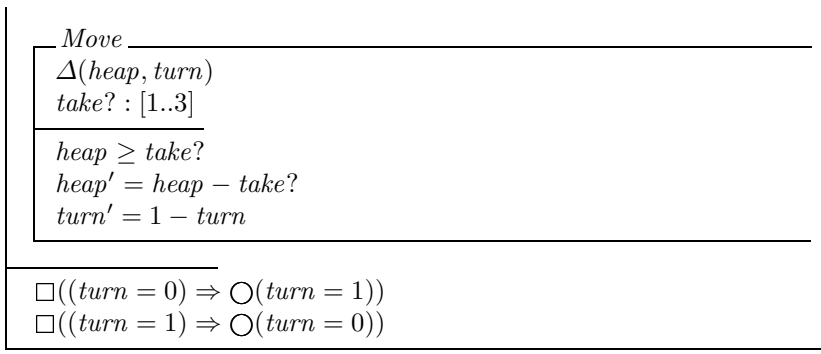
Object-Z [1] is a formal specification language which extends Z [7] by introducing object-orientation. Both are based upon set theory and first order predicate logic. Specification written in those languages include two parts: the system state and operations on states. Operations are constrained by pre and post conditions. Specification of an operation is enclosed in a definition scope called schema. Object-Z allows to group portions of the state and related operations in a class schema. It allows modularity, composition and inheritance. We propose an example of an Object-Z class based upon the *"heap of matches"* game. Two players, identified as player 0 and player 1, alternate in taking matches from a heap. The player whose turn has arrived decides whether he takes one, two or three matches. The player who takes the last match is the game's looser. The Object-Z description of the game class is the following:

Game

$heap : \mathbb{N}$
$turn : [0..1]$

INIT

$heap = 46$
$turn = 0$



Variables *heap* and *turn* specify respectively the number of matches in the heap and the current player identity. States are seen as bindings of model variables to values: the state of an object of the *Game* class is determined by the value bound to variables *heap* and *turn*. All $Object - Z$ classes include a schema named *INIT* to specify the set of initial states of the class. The *Move* schema specifies the class operation, a game move, and illustrates some notational conventions of $Object - Z$. The Δ statement indicates the portion of the state that is modified by the operation. Schemas that define operations are splitted in two zones by a short horizontal line. The upper zone specifies the portion of state involved in the operation and the operation parameters. The lower zone is occupied by predicates that express operation conditions. Another notational convention refers to the so-called *before/after* predicates. Primed variable names denote the value to which the variable is bound after the operation, unprimed variable names denote the value to which the variable was bound before the operation. Consequently, predicates without primed variables can be seen as operation preconditions and predicates with mixed primed and unprimed variables describe the effect of the operation on the state (i.e., the variable binding). Finally, a variable name followed by the question mark (?) distinguishes an operation's input parameter and a variable name followed by the exclamation mark (!) distinguishes an operation's output parameter. As for operation schemata, the class schema can be separated in two zones by a short horizontal line. The lower zone, if present, includes one or more temporal logic formulas, the history invariants. As we will examine in the sequel, history invariants allow to express liveness and safety constraints on the computations an object of the class can undergo, see [6] for a discussion on $Object - Z$ classes and history invariants. In our example, the given invariants state that game players should alternate in executing game moves. From the given specification of *Move* operation, this constraint is satisfied by all the possible computations of any instance of the game and the history invariants could be omitted.

2.2 Transition Systems

We intend to use transition systems to provide a semantics to classes of $Object - Z$. The intended meaning or "type" of an Object-Z class is the set of compu-

tations that objects of the class can undergo. A transition system is a compact representation of a set of computations and therefore appears as an adequate representation for the semantics of an Object-Z class. Furthermore, a number of verification techniques have been defined to check transition system models against temporal logic formulas.

A transition system $S =< \mathcal{V}, \Sigma, \mathcal{T}_e, \mathcal{T}_c, \Theta >$ includes a set $\mathcal{V}$ of typed variables, a set Σ of states, a set $\mathcal{T}_e$ of elementary transitions, a set $\mathcal{T}_c$ of compound transitions and an initial conditions Θ. Variables $\mathcal{V} = \mathcal{V}_i \cup \mathcal{V}_o \cup \mathcal{V}_p$ group in three classes: input ($\mathcal{V}_i$), output ($\mathcal{V}_o$) and private ($\mathcal{V}_p$). We assume that all common types and most of the classical type constructors are available, as it is the case for STeP. A state $s \in \Sigma$ is a valuation of the private variables. As for Z specifications, the state is determined by the value to which each variable is bound. If $v \in \mathcal{V}_p$ is a variable, we will note $s[v]$ the value bound to v in state s. Naturally, state s considered as a valuation function can be inductively extended to expressions of any kind. Particularly the initial condition Θ is a predicate such that, s_0 is an initial state if and only if $s_0[\Theta] = \text{true}$. We say that s_0 satisfies Θ, which is noted $s_0 \vDash \Theta$.

The sets $\mathcal{T}_e$ and $\mathcal{T}_c$ contain respectively the elementary and the compound transitions. Both represent state changes, i.e., functions of the kind: $\Sigma \rightarrow \mathbb{P}\,\Sigma$.

Generally, an elementary transition $\tau \in \mathcal{T}_e$ is described by its transition relation ρ_τ, i.e., a conjunction of predicates. As for the Z notation, we adopt the *before/after* convention to write those predicates. Among many possible general forms for a transition relation, we adopt the following:

$$\rho_\tau = \mathit{Pre}(\tau) \wedge (v_1' = \mathit{Exp}_1) \wedge \cdots \wedge (v_n' = \mathit{Exp}_n)$$

where $\mathit{Pre}(\tau)$, a predicate with only unprimed variables, is the transition's precondition: s is a source state for τ if and only if $s \vDash \mathit{Pre}(\tau)$. For each conjoint $(v_i' = \mathit{Exp}_i)$, $v_i \in \mathcal{V}_p \cup \mathcal{V}_o$ and Exp_i is an expression without primed variables. The set $\{v_1, \cdots, v_n\} \subseteq \mathcal{V}_p \cup \mathcal{V}_o$ is the subset of variables which change value as a consequence of the transition triggering. Finally, in the case of elementary transitions, unless otherwise stated, we assume that for each $v \notin \{v_1, \cdots, v_n\}$, there is an unwritten conjoint $(v' = v)$, meaning that variables that do not appear in $\rho(\tau)$ are supposed to be left unchanged by the transition.

Compound transitions are subsets of elementary transitions noted $[\tau_1;\ \cdots;\ \tau_n]$. Compound transitions should be considered as atomic: components of compound transitions do not interleave with other transitions of S. We introduce compound transitions to account for the atomicity of Object-Z class operations, as will be seen in section 3.1. We note $\mathcal{V}'(\rho_\tau)$ the set of primed variables of transition relation ρ_τ. A compound transition $[\tau_1;\ \cdots;\ \tau_n]$ is coherent if and only if:

$$\forall\, \tau_i, \tau_j \in [\tau_1, \cdots, \tau_n] \bullet i \neq j \Rightarrow \mathcal{V}'(\rho_{\tau_i}) \cap \mathcal{V}'(\rho_{\tau_j}) = \varnothing$$

A computation of transition system $S =< \mathcal{V}, \Sigma, \mathcal{T}_e, \mathcal{T}_c, \Theta >$ is a sequence

$$\sigma : s_0, s_1, \cdots, s_j, \cdots$$

of states such that $s_0 \vDash \Theta$ and for every $s_i, i \geq 0$, at least one of the following possibilities is verified:

• There is a $\tau \in \mathcal{T}_e$ such that $s_i \vDash Pre(\tau)$ and $s_{i+1} \in \tau(s_i)$.
• There is a coherent compound transition $[\tau_1; \cdots; \tau_n] \in \mathcal{T}_c$ that verifies what follows. Let $\mathcal{E} = \{\tau_1^e, \cdots, \tau_m^e\}$ be the set of components of τ_c enabled in s_i (i.e., for all j, $s_i \vDash Pre(\tau_j^e)$), then $\mathcal{E} \neq \varnothing$ and $s_{i+1} \in \tau_1^e(s_i) \cap \cdots \cap \tau_m^e(s_i)$.
We note $\mathcal{C}(S)$ the set of all computations of transition system S.

3 Verification Technique

3.1 From Object-Z Classes to Transition Systems

A class specification written in Object-Z can bee seen as the description of an abstract machine. Objects of the class are instances capable of producing computations of the machine, as sequences of state changes caused by operations. The aim of the proposed transformation is to obtain, from the definition of the class, written in Object-Z, a compact representation of the set of computations an object of the class can produce. This representation is intended to take the form of a transition system. The intended result of the transformation is a triple $(S, \mathcal{A}, \mathcal{H})$ where $S =< \mathcal{V}, \Sigma, \mathcal{T}_e, \mathcal{T}_c, \Theta >$ is a transition system as defined in section 2.2, $\mathcal{A}$ and $\mathcal{H}$ are sets of linear temporal logic (LTL) formulas called respectively axioms and history invariants. A simple *Object-Z* class definition has the following elements:

ClassName
type definitions
constant definitions
state schema
initial state schema
operations
---
history invariants

An *Object-Z* class named cl is characterized by the following sets. $Attr(cl)$ is a set of class attributes, declared in the *state schema*, $Param(cl)$ is a set of operation parameters. Both are subsets of a set of identifiers such that $Attr(cl) \cap Param(cl) = \varnothing$.
$State(cl) \subseteq Attr(cl) \nrightarrow Val$ is the set of class states, a subset of the finite partial functions from attributes to values. The set Val contains all possible values of attributes of any type. Finally, $Op(cl)$ is the set of class operations. We introduce the auxiliary functions $\pi_i : Op(cl) \rightarrow \mathbb{P}\, Param(cl)$ and $\pi_o : Op(cl) \rightarrow \mathbb{P}\, Param(cl)$ that give respectively the set of input and output parameters of an operation.
We explain now the translation of a simple *Object-Z* class named cl towards a model $(S^{cl}, \mathcal{A}^{cl}, \mathcal{H}^{cl})$, with $S^{cl} =< \mathcal{V}^{cl}, \Sigma^{cl}, \mathcal{T}_e^{cl}, \mathcal{T}_c^{cl}, \Theta^c l >$ and illustrate it with the *heap of matches* game. We state that:

$$\mathcal{V}_p^{cl} = Attr(cl),\ \mathcal{V}_i^{cl} = \bigcup_{o \in Op(cl)} \pi_i(o),\ \mathcal{V}_o^{cl} = \bigcup_{o \in Op(cl)} \pi_o(o)$$

so that, for the *Game* class we have $\mathcal{V}_p^{Game} = \{heap, turn\}$, $\mathcal{V}_i^{Game} = \{take?\}$ and $\mathcal{V}_o^{Game} = \varnothing$.
State schema of the form $[declaration\ part \mid Ax_1; \cdots; Ax_n]$ declares attributes and, optionally, states a list of axioms relative to declared attributes. Axioms $Ax_1, \cdots, Ax_n$ are first order predicates on the attributes defined in the declaration part. We state that:

$$\mathcal{A}^{cl} = \{Ax_1, \cdots, Ax_n\}$$

The general form of the initial state schema is $INIT \mathrel{\widehat{=}} [Pr_1; \cdots; Pr_m]$ where the $Pr_i (i \in [1..m])$ are first order predicates on the class attributes. We state that:

$$\Theta^{cl} = Pr_1 \wedge \cdots \wedge Pr_m$$

so that, for the *Game* class we have $\Theta^{Game} = (heap = 46) \wedge (turn = 0)$.
The class operations $OperationName \mathrel{\widehat{=}} [declaration\ part \mid predicate\ list]$ are the portion of the specification which is translated towards the transitions of S^{cl}. In many cases, an operation gives rise to a set of elementary transitions. Let $\mathcal{T}^o = \{\tau_1, \cdots, \tau_i\}$ be the set of elementary transitions resulting from operation $o \in Op(cl)$. We distinguish the following cases:
• if $\mathcal{T}^o$ is a singleton (i.e., $\mathcal{T}^o = \{\tau\}$), then $\tau \in \mathcal{T}_e^{cl}$.
• if $\mathcal{T}^o = \{\tau_1, \cdots, \tau_n\}$, with $n > 1$ then let $\tau_c = [\tau_1; \cdots; \tau_n] \in \mathcal{T}_c^{cl}$, provided that τ_c is coherent.
Given the richness of the Object-Z language, a detailed definition of all the translation rules from operations into transitions is a difficult task. We will illustrate some aspects of the translation principles through the *Game* example and also through the case study. Operation *Move* of the *Game* class gives rise to transition τ^{Move} as follows. The predicate with unprimed attributes determines the precondition: $Pre(\tau^{Move}) = (heap \geq take?)$. The predicates with mixed unprimed and primed attributes determine the rest of the transition relation. In the case of the *Game* example, the transition relation is straightforward:

$$\rho_{\tau^{Move}} = (heap \geq take?) \wedge (heap' = heap - take?) \wedge (turn' = 1 - turn)$$

and we have: $\mathcal{T}^{Game} = \mathcal{T}^{Move} = \{\tau^{Move}\}$.
Finally, the *history invariants* part of the class specification contains a list of LTL formulas $H_1, \cdots, H_p$. We state:

$$\mathcal{H}^{cl} = \{H_1, \cdots, H_p\}$$

So that we have $\mathcal{H}^{Game} = \{\Box((turn = 0) \Rightarrow \bigcirc(turn = 1)), \Box((turn = 1) \Rightarrow \bigcirc(turn = 0))\}$. Operations that translate to compound transitions are typically those which are specified by quantified formulas. In the case study we present rules to obtain compound transitions from operation specifications in which variables bound by quantifiers have finite types, such as intervals of natural integers.

3.2 Validity, Satisfiability, and Consistency

We present now the meaning of sets $\mathcal{A}$ and $\mathcal{H}$ that have been introduced in section 3.1. We use the fact that computations of transition systems can be models of LTL formulas. Mathematical logic gives a precise meaning to the word *model*, by defining the satisfaction relation between *"universe of discourse"* entities (models) and logic formulas. Let us begin by a quick remind of LTL formulas and their satisfaction relation.
LTL extends predicate calculus with new logical operators such as $\bigcirc$, $\mathcal{U}$, $\Box$, $\Diamond$, called the temporal operators. If F_1 and F_2 are well-formed formulas so are $\bigcirc F_1$, $F_1 \mathcal{U} F_2$, $\Box F_1$ and $\Diamond F_1$. State formulas are formulas without temporal operators. State-quantified formulas are formulas such that temporal operators do not appear in the scope of quantifiers.
The natural model for LTL formulas on a set $V = \{v_1, \cdots, v_n\}$ of variables are the so called Chains, i.e., sequences of valuations of V noted:

$$\sigma : s_0, \cdots, s_j, \cdots$$

with, for all j, $s_j : V \rightarrow Dom(V)$, where $Dom(V)$ denotes the domain where all variables of V take their values. We note $s[v]$ the value of variable $v \in V$ in state s. The satisfaction relation ($\vDash$) is defined as follows. If F is a state formula, then $(\sigma, j) \vDash F$, i.e σ satisfies F at position j, if and only if $s_j \vDash F$ (i.e., $s_j[F] = \text{true}$). If F_1 and F_2 are temporal formulas, the relation $\vDash$ is defined inductively: $(\sigma, j) \vDash \bigcirc F_1$ if and only if $(\sigma, j+1) \vDash F_1$; $(\sigma, j) \vDash F_1 \mathcal{U} F_2$ if and only if there exists $k \geq j$ such that $(\sigma, k) \vDash F_2$ and for all i such that $j \leq i < k$, $(\sigma, i) \vDash F_1$; $(\sigma, j) \vDash \Diamond F_1$ if and only if $(\sigma, j) \vDash \text{true} \mathcal{U} F_1$; $(\sigma, j) \vDash \Box F_1$ if and only if $(\sigma, j) \vDash \neg \Diamond \neg F_1$. The set of chains that satisfy an LTL formula F is noted $Sat(F)$.
Given an *Object-Z* class cl and its associated semantic entity $(S^{cl}, \mathcal{A}^{cl}, \mathcal{H}^{cl})$, the set $\mathcal{A}^{cl} = \{A_1, \cdots, A_m\}$ of axioms defines the state space of cl as the set:

$$\{s \in \Sigma^{cl} \mid s \vDash A_1 \wedge \cdots \wedge A_m\}$$

In other words, the axioms restrict the set Σ^{cl} of states of the transition system in order to ensure that $\mathcal{A}^{cl}$ is valid.
To define the meaning of the history invariant set, let us remind section 2.2, where we defined the set $\mathcal{C}(S)$ of computations of a transition system. We state that S^{cl} is consistent, with the set $\mathcal{H}^{cl} = \{H_1, \cdots H_n\}$ of history invariants if $Sat(H_1 \wedge \cdots \wedge H_n) \cap \mathcal{C}(S^{cl}) \neq \varnothing$. In other words, $\mathcal{H}^{cl}$ is expected to be satisfiable relatively to S^{cl}.

4 Case Study

4.1 The Radio-Mobile Network Example

In order to illustrate the specification and verification approach, we choose an example from the Radio-Mobile Network (RMN) design problem. Radio-mobile

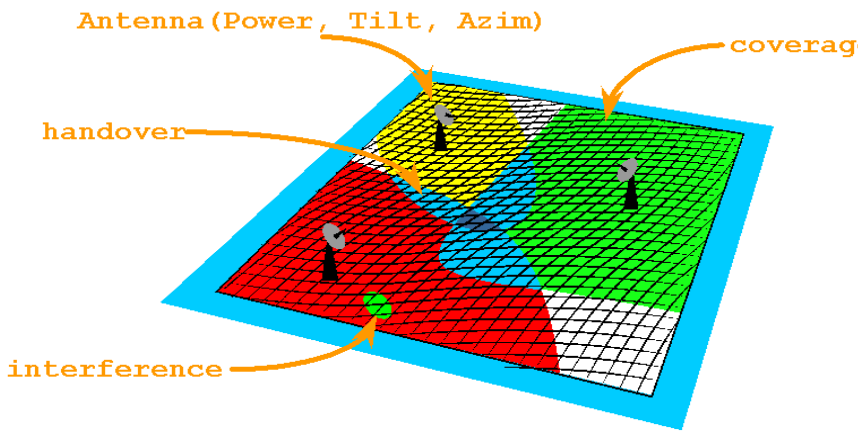


Fig. 1. Antenna parameter setting problem

communication relies on a network of antennae to cover a given area of territory, in order to ensure communication services such as mobile phones. The design of a RMN includes three main stages: antennae positioning, parameter setting and frequency allocation.

We are concerned here with the parameter setting stage: once the antennae locations have been determined in the positioning stage, the parameter setting problem deals with the issue of Quality of Service (QoS) optimization. The antenna parameter problem has yet been presented. Indeed, [3] gives an implemented solution to the problem, and in [2] we specify the problem and we simulate the specification.

For each antenna, three parameters need generally to be adjusted: emission power, tilt and azimuth. On the other hand, QoS includes several aspects, such as coverage, traffic, interference and handover. To optimize QoS, a trade-off must be found. For instance, coverage and handover ask for higher emission power while interference avoidance asks for lesser radio energy.

In order to solve the parameter setting problem, the area to cover is partitioned into a set of meshes whose size vary from 25 to 100 square meters, depending on the type of land, as illustrated by figure 1, which resumes the parameters setting problem.

For the sake of clarity, we restrict our example to the of emission power setting problem, of which we present now a formal description. Given a set $A = \{a_1, \cdots, a_n\}$ of antennae and a set $M = \{m_1, \cdots, m_k\}$ of meshes, the following functions define the radio-propagation upon the considered area. Function $Fade : M \times A \rightarrow \mathbb{R}$ allows to calculate the mesh-local variation of the field emitted by each antenna. Function $Power : A \rightarrow \mathbb{R}$ gives the radio-electric emission power of each antenna. Function $F : M \times A \rightarrow \mathbb{R}$ allows to calculate the field intensity in any mesh, due to each antenna emission. As a matter

of fact, for mesh m_i and antenna a_j, the field intensity can be calculated as $F(m_i, a_j) = f(Fade(m_i, a_j), Power(a_j))$, for a given function f. In this example we assume that f is simply the product between emission power and attenuation:

$$F(m_i, a_j) = Fade(m_i, a_j) * Power(a_j)$$

4.2 Specification

The specification of the antennae parameter setting problem is structured in three classes. The first specifies the terrain upon which antennae are situated, the second specifies an antennae and the latter specifies the system as a whole. We suppose that antennae and meshes numbers are fixed and known. They are specified by the following schema.
Types *MESHID* and *ANTENNAID* allow to identify each mesh and each antenna. Constant function Fade gives, for each antenna, the field attenuation in any mesh and thereby allows to calculate field intensity in any mesh, due to any antenna. We adopt the view of identifying discrete emission power levels, as defined by type *POWERLEVEL*. The only state variable is *Field* which, for any mesh, gives a field intensity vector indexed by the set of antennae identifiers, the contribution of each antenna to the mesh's field. Operation *ChangeField* allow to calculate the contribution of an antenna to the field after a change in antenna's emission power. Operation *MeasureField* returns, for a given antenna, the field it contributes to any mesh. Operation *DetectInterference* returns, for a given antenna, the indication of whether it interferes another antenna in any mesh.

Terrain

$n : \mathbb{N}$ [number of antennae]
$k : \mathbb{N}$ [number of meshes]

$n < k$

$MESHID == [1..k]$
$ANTENNAID == [1..n]$
$POWERLEVEL == [0..100]$

$Fade : MESHID \times ANTENNAID \rightarrow \mathbb{R}$

$\forall\, i : MESHID, j : ANTENNAID \bullet 0 \leq Fade(i, j) < 1$

$Field : MESHID \rightarrow (ANTENNAID \rightarrow \mathbb{R})$

INIT

$\forall\, m : MESHID;\ a : ANTENNAID \bullet Field(m)(a) = 0$

ChangeField

$\Delta Field$
$EmitedPower? : POWERLEVEL$
$AntId? : ANTENNAID$

$\forall\, m : MESHID \bullet$
$Field'(m) = Field(m) \oplus \{AntId? \mapsto Fade(m, AntId?) * EmitedPower?\}$

MeasureField

$AntId? : ANTENNAID$
$FieldMeasure! : MESHID \to \mathbb{R}$

$\forall\, m : MESHID \bullet FieldMeasure!(m) = Field(m)(AntId?)$

DetectInterference

$AntId? : ANTENNAID$
$Interfered! : MESHID \to \mathbb{B}$

$\forall\, m : MESHID \bullet Interfered!(m) =$
$\exists\, a : ANTENNAID \bullet a \neq AntId? \land Field(m, a) \geq gate_Q \land$
$\quad Field(m, AntId?) \geq gate_S$

An antenna is specified by its identity, emission power and coverage attributes and operations. Operation *SetCoverage* determines, from field intensity, the meshes covered by the antenna. Operation *SetPower* bases on interference to determine whether the antenna increases or decreases its emission power. Operation *Emit* outputs the antenna's emission power.

Antenna

$n : \mathbb{N}$ [number of antentennae]
$k : \mathbb{N}$ [number of meshes]

$n < k$

$MESHID == [1..k]$
$ANTENNAID == [1..n]$
$POWERLEVEL == [0..100]$

$AntId : ANTENNAID$
$EmissionPower : POWERLEVEL$
$Covered : MESHID \to \mathbb{B}$
$Coverage : \mathbb{N}$

INIT

$$EmissionPower = 0$$
$$Coverage = 0$$

SetCoverage

$$\Delta(Covered, Coverage)$$
$$Identity! : ANTENNAID$$
$$FieldMeasure? : MESHID \rightarrow \mathbb{R}$$

$$Identity! = AntId$$
$$\forall\, m : MESHID \bullet Covered'(m) = (FieldMeasure?(m) > gate_Q)$$
$$Coverage' = \#(Covered' \rhd \text{true})$$

SetPower

$$\Delta(EmissionPower)$$
$$Identity! : ANTENNAID$$
$$Interfered? : MESHID \rightarrow \mathbb{B}$$

$$Identity! = AntId$$
$$\#(Interfered? \rhd \text{true}) < I_{max} \land$$
$$\quad EmissionPower' = EmissionPower + \Delta_1 Power$$
$$\#(Interfered? \rhd \text{true}) \geq I_{max} \land$$
$$\quad EmissionPower' = EmissionPower - \Delta_2 Power$$

Emit

$$Identity! : ANTENNAID$$
$$Power! : POWERLEVEL$$

$$Identity! = AntId$$
$$Power! = EmissionPower$$

The system composed of antennae and meshes is specified by *CommunicationSystem* class. It includes an instance of the *Terrain* class and a set of instances of the *Antenna* class, indexed by the set of antenna identifiers.

CommunicationSystem

$$terrain : Terrain$$
$$antenna : ANTENNAID \rightarrow Antenna$$

$$\forall\, a : ANTENNAID \bullet antenna(a).AntId = a$$

INIT

$$terrain.INIT$$
$$\forall\, a : ANTENNAID \bullet antenna(a).INIT$$

SetAllPowers

$\Delta(antenna)$

$\forall\, a : ANTENNAID \bullet antenna(a).SetPower \parallel terrain.DetectInterference$

SetAllCoverages

$\Delta(antenna)$

$\forall\, a : ANTENNAID \bullet terrain.MeasureField \parallel antenna(a).SetCoverage$

SetField

$\Delta(terrain)$

$\forall\, a : ANTENNAID \bullet antenna(a).Emit \parallel terrain.ChangeField$

$\Box((op = SetAllPowers) \rightarrow \bigcirc(op = SetField))$
$\Box((op = SetField) \rightarrow \bigcirc(op = SetAllCoverages))$
$\Box((op = SetAllCoverage) \rightarrow \bigcirc(op = SetAllPowers))$

4.3 Verifying the Object-Z Specification

Obtaining the Transitions System We discuss now some components of the transition system that results from the Object-Z specification of the case study. We do not express formally in this work the rules that guide such a transformation. Rather, we give an informal presentation to justify the transitions resulting from some constructs frequently used in the case study. An important issue concerns the evaluation of compound transitions by evaluating sequentially their elementary component transitions. This can be made under some conditions that we present below.

Many assertions that specify the operations of our model are universally quantified formulas, with bound variables ranging in sets *MESHID* and *ANTENNAID*. The general idea to translate such assertions into transitions is as follows. With formula:

$$\forall\, i : [1..k] \bullet Pre_i \wedge (var_i' = Exp_i)$$

where Pre_i is a precondition, var_i is a variable name and Exp_i is an expression, we associate the compound transition $[\tau_1;\ \cdots\ ;\ \tau_k]$ such that for all $i \in [1..k]$, the transition relation is:

$$\rho(\tau_i) = Pre_i \wedge (var_i' = Exp_i)$$

If for all $i, j \in [1..k]$ we have $i \neq j \Rightarrow var_i \neq var_j$ then we have a coherent transition. Moreover, this compound transition can be evaluated sequentially if neither Pre_i nor Exp_i contain any variable other than var_i.

Another Z construct frequently used in our specification is the range restriction operator ($\rhd$). We used it when, given a predicate on *Mesh*, we had to count

the domain elements that mapped to true. A general rule to translate such a construct towards a set of transitions, is the following. Given a predicate $Pred : [1..k] \rightarrow \mathbb{B}$, with formula:

$$count' = \#(Pred(i) \rhd true)$$

we associate the compound transition $[(count' = 0);\ \tau_1;\ \cdots;\ \tau_k]$ such that for all $i \in [1..k]$, the transition relation is:

$$\rho(\tau_i) = Pred(i) \wedge (count' = count + 1)$$

Indeed, this is not a coherent compound transition but we can obtain the expected result by evaluating sequentially each one of the component elementary transitions.

We dealt with some other aspects of the Object-Z specification, such as aggregation, before translation towards transition systems. As a matter of fact, we inspired from the approach to class aggregation presented in [6]. As an example, the *CommunicationSystem* class results from the aggregation of classes *Terrain* and *Antenna*. Operations of the aggregate class were defined by applying operator $\|$ to operations of component classes. To obtain the translation of those operations towards transitions of the transition system, we first applied schema transformations due to the $\|$ operator of [6]. As an example, consider the *SetField* operation, which refers to operation $antenna(a).Emit \ \| \ terrain.ChangeField$. We have:

$antenna(a).Emit \ \| \ terrain.ChangeField$

$\Delta(terrain)$

$ChangeField.AntId? = Emit.Identity!$
$Terrain.EmitedPower? = Emit.Power!$
$terrain \ \underline{Terrain} :: \underline{ChangeField} \ terrain'$

where, according to operation $antenna(a).Emit$, we have:

$$Identity! = antenna(a).AntId$$

and

$$Power! = antenna(a).EmissionPower$$

On the other hand, remind that relation $terrain \ \underline{Terrain} :: \underline{ChangeField} \ terrain'$ states that *terrain* and $terrain'$ are related by the aggregated operation exactly as they are related by operation *ChangeField* of class *Terrain*. If we apply the equalities and the definition of the $terrain \ \underline{Terrain} :: \underline{ChangeField} \ terrain'$ relation, we obtain:

$$antenna(a).Emit \parallel terrain.ChangeField$$
$$\Delta(terrain)$$
$$\forall\, m : MESHID \bullet Field'(m) = Field(m) \oplus \{antenna(a).AntId \mapsto Fade(m, antenna(a).AntId) * antenna(a).EmissionPower\}$$

which is the predicate to be translated towards transitions.

Verifying with STeP: Some Implementation Details The use of STeP as a verification tool requires the expression of the resulting transition system according to the STeP transitions syntax. As the later do not include the notion of compound transitions defined in section 2.2, we defined an implementations of compound transitions based on the sequential evaluation of elementary transitions components. To ensure atomicity of compound transitions we used lock variables. We factorized sets of transitions by implementing loop constructs. Finally, Functions on domain *MESHID* (respectively *ANTENNAID*) were implemented as arrays indexed by $[1..k]$ (respectively $[1..n]$) and functions on domain $MESHID \times ANTENNAID$ were implemented as arrays indexed by $[1..k] \times [1..n]$. As an example consider operation *SetCoverage* of class *Antenna*, which produced the following transition relations:

$$\rho_1 = (lock = FREE) \wedge (index' = 1) \wedge (Coverage' = 0) \wedge (lock' = TAKEN)$$
$$\rho_2 = (index \geq 1) \wedge (index \leq k) \wedge (FieldMeasure?[index] \geq gate_Q) \wedge$$
$$(Covered'[index] = \text{true}) \wedge (Coverage' = Coverage + 1) \wedge (index' = index + 1)$$
$$\rho_3 = (index \geq 1) \wedge (index \leq k) \wedge (FieldMeasure?[index] < gate_Q) \wedge$$
$$(Covered'[index] = \text{false}) \wedge (index' = index + 1)$$
$$\rho_4 = (index > k) \wedge (index' = 0) \wedge (lock' = FREE)$$

We used the STeP model-checker to verify the satisfiability of the history invariants of class *CommunicationSystem*, on a simple communication system composed of two antennae and ten meshes. The STeP model-checker proves or refutes validity of LTL formulas relatively to a transition system. to establish the satisfiability of history invariant H one must actually establish that $\neg\, H$ is not valid. Concerning our case study, we checked separately each one of the three temporal formulas of class *CommunicationSystem*. For instance, the satisfiability of the first formula was checked by establishing that formula $\neg \Diamond \neg ((op = SetAllPowers) \rightarrow \bigcirc(op = SetField))$ was not valid.
We first verified some properties of separated classes. For the *Antenna* class, we verified the satisfiability of formula $F_1 = \Diamond(Coverage \geq Cov_{min})$ with Cov_{min} being a minimal number of covered meshes. This property represents the possibility of an important result to be obtained, i.e., an antenna covering at least a minimal number of meshes. As already mentioned, the STeP model-checker proves or refutes validity of LTL formulas relatively to a transition system. Consequently, STeP was asked to verify formula $\Box(Coverage < Cov_{min})$. A counterexample was rapidly established by STeP. The number of states visited by the

model-checker was 48. Next we verified a safety property, expressed by formula $F_2 = \Box((Coverage \leq k) \Rightarrow (EmmissionPower < 100))$, to mean that even when all meshes are covered by an antenna, the maximal power level is not exceded. This property was refuted by STeP with a counterexample, thereby motivating a change in the *Antenna* specification. The modified version satisfied F_2, but verifying this formula took considerable amounts of time, visiting more than 200000 states.

5 Conclusion

Formalisms like Z and Object-Z include in a unique specification language two important conceptual domains. On one side, by adopting the first order predicate notation, they belong to the *specify with logic* realm. On the other side, as they describe system state and operations on states, Object-Z also relates to more operational styles of specification. In this work on verification of Object-Z specifications we propose some rules to transform an Object-Z class into a transition system, in order to use verification tools such as the STeP environment. Rather than general rules, we present the approach by applying it to a case-study. The proposed informal rules transform operations of the class in sets of simple transitions grouped in indivisible compound transitions to represent operations of the class. Future research should concentrate on a better understanding of problems related to the transformations from Object-Z operations towards compound transitions, that preserve! the properties of operations expressed with first order predicates. Formal presentation of a set of well defined transformation rules is also necessary to consolidate the approach.

References

1. Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z specification language. Technical report, Software Verification Research Center, Department of Computer Science, University of Queensland, AUSTRALIA, 1991.
2. V. Hilaire, T. Lissajoux, and A. Koukam. AgentCharts: An Operational Model For Multi-Agent Systems. In *International Conference on Advanced Computer Systems ACS'98*, 1998.
3. T. Lissajoux, V. Hilaire, A. Koukam, and A. Caminada. Genetic algorithms as prototyping tools for multi-agent systems: Application to the antenna parameter setting problem. In Springer Verlag, editor, *Lecture Note in Artificial Intelligence*, number 1437 in LNAI, 1998.
4. Z. Manna, N. Bjoerner, A. Browne, and E. Chang. STeP: The Stanford Temporal Prover. *Lecture Notes in Computer Science*, 915:793–??, 1995.
5. Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.
6. Graeme Paul Smith. *An Object-Oriented Approach To Formal Specification.* PhD thesis, University of Queensland, 1992.
7. J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall, 1992.