USING FORMAL SPECIFICATION TO VALIDATE A HUMAN RESOURCE INFORMATION SYSTEM

Rohidah Maskuri
Sepang Institute of Technology
Level 5 Klang Parade, 2112 Jalan Meru
Klang, Selangor
Malaysia
Tel.: 603-3430628
email: rohidah@sit.edu.my

Abdullah Mohd. Zin
Faculty Technology and Information Science
Universiti Kebangsaan Malaysia
43600 Bangi, Selangor
Malaysia
Tel.: 603-8296179
email: amz@suns1.ftsm.ukm.my

ABSTRACT

This paper contains the analysis of the Z specification of a human resource information system. It aims to show the strength of the formal methods in analysing and detecting errors after the implementation phase.

Keywords: ***Prototyping, Formal Methods, Initialisation Theorems, Preconditions***

1.0 INTRODUCTION

Prototyping is one of the traditional, yet most popular methodologies that is widely being practised in software development. This is due to one main factor, it allows the product to be delivered to the customer in the shortest time scale. The prototyping paradigm begins with the requirements gathering. A meeting between developer and customer is arranged to define the objectives as well as to identify the requirements of the software.

Following the meeting, a model which focuses on a representation of those aspects of the software that will be visible to the customer such as the format of the input and output will be designed. The quick design then leads to the construction of a prototype. At this point, the customer may decide that the requirements are wrong, or that they have changed, or that new features should be added. Iteration occurs as the prototype is tuned to satisfy the needs of the customer [1].

One main advantage of this approach is the quick turnaround in designing and building the prototypes. Customers are able to preview the system even before the system is fully implemented. No doubt that prototyping is preferred by most of the customers because of its fast delivery, yet it can be problematic for certain reasons [2].

First, customers are normally mislead into regarding the first prototype as the end result. The first prototype presented to the customer is just the first phase of an iterative system development process.

Second, for the sake of demonstration, developers tend to design the first prototype using inappropriate language and platform. However, the nature of iteration practised in this approach always causes this to be ignored. This affects the overall performance of the system in the long run.

Third, and the most significant disadvantage is that the requirements for the system are not fully captured since the system is built during the implementation phase. Thus, the chances of producing errors are high. Boehm discovered that over 60 percent of errors encountered in the systems during operations were due to shortcomings in the specifications [3].

A formal method is another useful methodology in software development due to its role to produce a defect-free software. By applying a rigorous mathematical notation to specify, develop, and verify a computer-based system, this approach certainly provide a promising method to increase confidence in developing a system prior to system implementation [4]. This mechanism which makes use of mathematical analysis is very useful in overcoming difficulties such as ambiguity, incompleteness, and inconsistency when used during development. Thus, it is not too outrageous to claim that formal methods model does offer the promise of defect-free software.

Formal specification is normally used to guide the software development process. However, this paper explores a new approach towards formal methods, that is the use of formal methods in analysing existing systems. In particular, this paper attempts to show that formal methods can be used as a verification tool, that is to find and correct errors after the system implementation.

2.0 METHODOLOGY

To study the effectiveness of formal methods in verifying a system, a human resource information system of one institute of higher learning was studied. The main task of the system is to maintain the record of each staff working in the institute. After considering the constraints and requirements of the system, a formal specification for the system is produced by using Z notations [5]. The Z formal specification of the system is described by Mohd. Zin and Maskuri [6]. Below is the summary of the methodology.

- study the human resource information system
- develop formal specification for the system

- prove the formal specification of the system.

The first two items above are covered in the paper by Mohd. Zin and Maskuri [6] while the third one is the main focus of this paper.

3.0 VALIDATION PROCESS

A formal specification is valid if it is self-consistent and complete. The self-consistency and completeness of a Z formal specification is determined by proving the following three things [7]:

- Proof of Initialisation Theorem
- Simplification of Preconditions
- Proof of Properties

The Proof of Initialisation Theorem and Simplification of Preconditions are two standard checks that are carried out for any state-based specification. Initialisation Theorem is a theorem concerning the existence of at least one suitable initial state. It takes the form of

$$\vdash \exists \; Schema' \bullet InitSchema$$

This theorem states that there really is a *Schema'* which satisfies the requirements of *InitSchema.* An initial state of the system should always be specified. The purpose of the initial state is to demonstrate that at least one beginning state exists before any operation on the state can be performed.

Unlike any other states, an initial state has one interesting property, it does not have any 'before state' which means that there is no operation applicable before the existence of this state. However, an initial state will produce an 'after state' once the first operation is performed on it.

The purpose of the second check, the precondition calculation is to examine that the operation is valid. There must be at least one 'before state' in which the operation is applicable. To make an operation applicable, a combination of 'before state' and a set of inputs must exist. There must also exist a combination of 'after state' and set of outputs that satisfies the relationship amongst all the variable involved.

The precondition for an operation schema *Op* is denoted in Z as

$$pre\ Op$$

In developing the precondition of a schema, all the output variables and after state variables are existentially quantified and allocated under the predicate part. For example, if *Op* represents an operation, *pre Op* is defined to mean

$$\exists \; State' \, ; \, Outputs! \bullet Op$$

where *State*' is the after state of the system for which *Op* is defined and *Outputs!* is the set of declarations of the output variables of *Op*.

In many cases, the precondition can be simplified to give a shorter but logically equivalent statement.

Having proven the first two levels, the last type of proof, the Proof of Properties is the formal checking on the informal properties and requirements in the system that need to be captured.

4.0 VALIDATING A HUMAN RESOURCE INFORMATION SYSTEMS

This section will practically apply the above validation process to prove a human resource information system mentioned earlier. In order to do that, this section will be divided into three parts. The first part is to check the Initial State Theorem for the state schemas of the Z formal specification while the second part is to calculate the preconditions of all the operation schemas of the specification. The last section involves in analysing and verifying the properties of the system.

4.1 Initial State Theorem

This section is divided into two parts. The first section will generate all the initial state schemas for the specification while the second part will involve proving the produced initial state schemas.

There are five state schemas for the system: *Person, Address, Independence, Beneficiary* and *Salary*. The prefix *Init* to the schema name is used to denote an initial state schema. For example, *InitPerson* is the initial state for the schema *Person*.

The initial state for all the schemas are given in Fig 1.

Initial state schemas are validated by proving their corresponding initial state theorem. Since the steps taken to prove all the initial state theorems are almost the same, this section will only show the steps taken to prove the initial state theorem for *InitPerson*. A summary of the final steps for the rest of the schemas will be shown at the end of this section.

The corresponding initial state theorem for the schema *Person* is given below:

$$\vdash \exists \; Person' \bullet InitPerson$$

The statement of the theorem can be expanded and simplified. The simplification of this theorem is shown in Fig. 2.

(a)

InitPerson

Person'

dom *name'* = **dom** *title'* = **dom** *sex'* = **dom** *race'* = **dom** *religion'* = **dom** *origin'* = **dom** *pensionno'* = **dom** *telext'* = **dom** *status'* = **dom** *dob'* = **dom** *icno'* = **dom** *newicno'* = **dom** *citizenship'* = **dom** *state'* = **dom** *maritalstatus'* = **dom** *pensiondate'* = **dom** *dobno'* = **dom** *disability'* = **dom** *taxno'* = **dom** *epfstatus'* = **dom** *socsono'* = **dom** *epfno'* = *staff'* = Ø

(b)

InitAddress

Address'

dom *address1'* = **dom** *address2'* = **dom** *address3'* = **dom** *postcode'* = **dom** *town'* = **dom** *state'* = **dom** *telephone'* = **dom** *telext'* = Ø

(c)

InitIndependence

AllIndependence'

dom *child'* = **dom** *spouse'* = **dom** *parents'* = Ø

(d)

InitBeneficiary

Beneficiary'

dom *address1benef'* = **dom** *address2benef'* = **dom** *address3benef'* = **dom** *postcodebenef'* = **dom** *townbenef'* = **dom** *statebenef'* = **dom** *phonebenef'* = **dom** *offphonebenef'* = **dom** *sambtelwaris'* = Ø

(e)

InitSalary

Salary'

dom *bankcode'* = **dom** *bankbranch'* = **dom** *accountno'* = Ø

Fig. 1: Initial State Schemas

$$
\begin{array}{l}
\vdash \exists \; name' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40name \\
title' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40title \\
sex' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40sex \\
race' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40race \\
religion' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40religion \\
origin' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40origin \\
pensionno' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01pensionno \\
dob' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40dob \\
icno : kk01personno \mathrel{+\!\!\!\rightarrow} kk40icno \\
newicno' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40newicno \\
citizenship : kk01personno \mathrel{+\!\!\!\rightarrow} kk40citizenship \\
state' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01state \\
maritalstatus' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40maritalstatus \\
pensiondate' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01pensiondate \\
dobno' : kk01personno \mathrel{+\!\!\!\rightarrow} kk40dobno \\
disablecode' : kk01personno \mathrel{+\!\!\!\rightarrow} kk02disablecode \\
taxno' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01taxno \\
epfstatus' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01epfstatus \\
socsono' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01socsono \\
socsotype' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01socsotype \\
epfno' : kk01personno \mathrel{+\!\!\!\rightarrow} kk01epfno \\
staff' : \mathbb{P}\, kk01person \; \bullet \\
\mathbf{dom}\; name' = \mathbf{dom}\; sex' = \mathbf{dom}\; title' = \mathbf{dom}\; race' = \mathbf{dom}\; religion' = \mathbf{dom}\; origin' = \\
\mathbf{dom}\; pensionno' = \mathbf{dom}\; telext' = \mathbf{dom}\; ukmstatus' = \mathbf{dom}\; dob' = \mathbf{dom}\; icno' = \mathbf{dom} \\
newicno' = \mathbf{dom}\; citizenship' = \mathbf{dom}\; state' = \mathbf{dom}\; maritalstatus' = \mathbf{dom} \\
pensiondate' = \mathbf{dom}\; dobno' = \mathbf{dom}\; disablecode' = \mathbf{dom}\; taxno' = \mathbf{dom}\; epfstatus' = \\
\mathbf{dom}\; socsono' = \mathbf{dom}\; socsotype' = \mathbf{dom}\; epfno' = staff' = \emptyset
\end{array}
$$

Fig. 2: Simplification of Schema *InitPerson*

Since all the after state variable share the common domain, the above can be further reduced to become,

$$staff' = \emptyset$$

The requirement of *Person'* is certainly met when *staff'* is set to empty. By using the same approach, it is also proven that all the other initial state theorems are true.

4.2 Precondition Calculation

As mentioned earlier, a formal specification is valid if it is self-consistent and complete. The consistency of an operation can be checked by calculating the precondition for that operation and also by checking that it agrees with our intuition. Thus, to complete this section, the expected preconditions for all available operations is presented. Following that, mathematical analysis will be used to calculate the precondition manually. The details of the steps taken in calculating every precondition are no shown due to the space constraint. Only the details for the schema D*isplayPerson* will be illustrated. The summary for all the simplified preconditions will be presented right after that.

In developing the precondition of a schema, all the output variable and after state variable are existentially quantified and allocated under the predicate part. As it progresses, One Point Rule will be used regularly to reduce predicates.

For the schema *DisplayPerson* the precondition is given in Fig. 3.

PreDisplayPerson

$$\begin{array}{l}
\mathit{name} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40name} \\
\mathit{title} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40title} \\
\mathit{sex} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40race} \\
\mathit{origin} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40origin} \\
\mathit{pensionno} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk01pensionno} \\
\mathit{telext} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk01telext} \\
\mathit{ukmstatus} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40status} \\
\mathit{dob} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40dob} \\
\mathit{icno} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40icno} \\
\mathit{newicno} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk01newicno} \\
\mathit{citizenship} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40citizenship} \\
\mathit{state} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk01state} \\
\mathit{maritalstatus} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk40maritalstatus} \\
\mathit{pensiondate} : \mathit{kk01personno} \mathrel{+\!\!\!\!\rightarrow} \mathit{kk01pensiondate} \\
\mathit{personno?} : \mathit{kk01personno} \\
\mathit{staff} : \mathbb{P}\ \mathit{kk01personno}
\end{array}$$

$$\begin{array}{l}
\exists\ \mathit{name!}:\mathit{kk40name};\ \mathit{title!}:\mathit{kk40title};\ \mathit{sex!}:\mathit{kk40sex};\ \mathit{origin!}:\mathit{kk40origin}; \\
\mathit{dob!}:\mathit{kk40dob};\ \mathit{pensionno!}:\mathit{kk01pensionno};\ \mathit{telext!}:\mathit{kk01telext}; \\
\mathit{ukmstatus!}:\mathit{kk40status};\ \mathit{icno!}:\mathit{kk40icno};\ \mathit{newicno!}:\mathit{kk01newicno}; \\
\mathit{citizenship!}:\mathit{kk40citizenship};\ \mathit{state!}:\mathit{kk01state};\ \mathit{ensiondate!}:\mathit{kk01pensiondate}; \\
\mathit{maritalstatus!}:\mathit{kk40maritalstatus} \bullet \\
\mathit{personno?} \in \mathit{staff} \wedge \\
\mathit{name!} = \mathit{name}(\mathit{personno?}) \wedge \\
\mathit{title!} = \mathit{title}(\mathit{personno?}) \wedge \\
\mathit{sex!} = \mathit{sex}(\mathit{personno?}) \wedge \\
\mathit{origin!} = \mathit{origin}(\mathit{personno?}) \wedge \\
\mathit{pensionno!} = \mathit{pensionno}(\mathit{personno?}) \wedge \\
\mathit{telext!} = \mathit{telext}(\mathit{personno?}) \wedge \\
\mathit{ukmstatus!} = \mathit{ukmstatus}(\mathit{personno?}) \wedge \\
\mathit{dob!} = \mathit{dob}(\mathit{personno?}) \wedge \\
\mathit{icno!} = \mathit{icno}(\mathit{personno?}) \wedge \\
\mathit{newicno!} = \mathit{newicno}(\mathit{personno?}) \wedge \\
\mathit{citizenship!} = \mathit{citizenship}(\mathit{personno?}) \wedge \\
\mathit{state!} = \mathit{state}(\mathit{personno?}) \wedge \\
\mathit{maritalstatus!} = \mathit{maritalstatus}(\mathit{personno?}) \wedge \\
\mathit{pensiondate!} = \mathit{pensiondate}(\mathit{personno?}) \wedge \\
\mathbf{dom}\ \mathit{name} = \mathbf{dom}\ \mathit{title} = \mathbf{dom}\ \mathit{sex} = \mathbf{dom}\ \mathit{origin} = \mathbf{dom}\ \mathit{pensionno} = \mathbf{dom}\ \mathit{telext} \\
= \mathbf{dom}\ \mathit{ukmstatus} = \mathbf{dom}\ \mathit{dob} = \mathbf{dom}\ \mathit{icno} = \mathbf{dom}\ \mathit{newicno} = \mathbf{dom}\ \mathit{citizenship} = \\
\mathbf{dom}\ \mathit{state} = \mathbf{dom}\ \mathit{maritalstatus} = \mathbf{dom}\ \mathit{pensiondate}
\end{array}$$

Fig. 3: Precondition for the Schema *DisplayPerson*

By the one point rule [5], all the variables under the existential quantifier can be eliminated to result in the next schema as in Fig. 4. Following the *PreDisplayPerson* is the summary for all the simplified preconditions for the operation schemas produced by Mohd. Zin and Maskuri [6].

PreDisplayPerson

name : kk01personno ⇸ *kk40name*
title : kk01personno ⇸ *kk40title*
sex : kk01personno ⇸ *kk40race*
origin : kk01personno ⇸ *kk40origin*
pensionno : kk01personno ⇸ *kk01pensionno*
telext : kk01personno ⇸ *kk01telext*
ukmstatus : kk01personno ⇸ *kk40status*
dob : kk01personno ⇸ *kk40dob*
icno : kk01personno ⇸ *kk40icno*
newicno : kk01personno ⇸ *kk01newicno*
citizenship : kk01personno ⇸ *kk40citizenship*
state : kk01personno ⇸ *kk01state*
maritalstatus : kk01personno ⇸ *kk40maritalstatus*
pensiondate : kk01personno ⇸ *kk01pensiondate*
personno? : kk01personno
staff : $\mathbb{P}$ *kk01personno*

---

personno? ∈ *staff* ∧
name! = name(personno?) ∧
title! = title(personno?) ∧
sex! = sex(personno?) ∧
origin! = origin(personno?) ∧
pensionno! = pensionno(personno?) ∧
telext! = telext(personno?) ∧
ukmstatus! = ukmstatus(personno?) ∧
dob! = dob(personno?) ∧
icno! = icno(personno?) ∧
newicno! = newicno(personno?) ∧
citizenship! = citizenship(personno?) ∧
state! = state(personno?) ∧
maritalstatus! = maritalstatus(personno?) ∧
pensiondate! = pensiondate(personno?) ∧
dom *name* = **dom** *title* = **dom** *sex* = **dom** *origin* = **dom** *pensionno* = **dom**
telext = **dom** *ukmstatus* = **dom** *dob* = **dom** *icno* = **dom** *newicno* = **dom**
citizenship = **dom** *state* = **dom** *maritalstatus* = **dom** *pensiondate*

Fig. 4: Simplified Precondition for Schema *DisplayPerson*

Table 1: Summary of the Simplified Preconditions for all the Operation Schemas

Operations	**Simplified Preconditions**
PreDisplayPerson	$personno? \in staff$ $name(personno?) \in kk40name$ $title(personno?) \in kk40title$ $sex(personno?) \in kk40sex$ $origin(personno?) \in kk40origin$ $pensionno(personno?) \in kk40pensionno$ $telext(personno?) \in kk40telext$ $ukmstatus(personno?) \in kk40status$ $dob(personno?) \in kk40dob$ $icno(personno?) \in kk40icno$ $newicno(personno?) \in kk01newicno$ $citizenship(personno?) \in kk40citizenship$ $state(personno?) \in kk01state$ $maritalstatus(personno?) \in kk40maritalstatus$ $pensiondate(personno?) \in kk40pensiondate$
PreDisplaySpouse	$personno? \in staff$ $spouse(personno?) \neq \emptyset$ $spouse(personno?) \in \mathbb{P}\ independence$
PreDisplayIndependence	$personno? \in staff$ $child(personno?) \neq \emptyset$ $spouse(personno?) \neq \emptyset$ $parent(personno?) \neq \emptyset$ $child(personno?) \in \mathbb{P}\ independence$ $parent(personno?) \in \mathbb{P}\ independence$ $spouse(personno?) \in \mathbb{P}\ independence$
PreUpdateStaff	$personno? \in staff$ $name \oplus \{personno? \mapsto name?\} \in kk01personno \nrightarrow kk40name$ $icno \oplus \{personno? \mapsto icno?\} \in kk01personno \nrightarrow kk40icno$ $title \oplus \{personno? \mapsto title?\} \in kk01personno \nrightarrow kk40title$ $sex \oplus \{personno? \mapsto sex?\} \in kk01personno \nrightarrow kk40sex$ $race \oplus \{personno? \mapsto race?\} \in kk01personno \nrightarrow kk01race$ $origin \oplus \{personno? \mapsto origin?\} \in kk01personno \nrightarrow kk40origin$ $religion \oplus \{personno? \mapsto religion?\} \in kk01personno \nrightarrow kk40religion$ $maritalstatusn \oplus \{personno? \mapsto maritalstatus?\} \in kk01personno \nrightarrow kk40maritalstatus$ $newicno \oplus \{personno? \mapsto newicno'?\} \in kk01personno \nrightarrow kk01newicno$ $citizenship \oplus \{personno? \mapsto citizenship?\} \in kk01personno \nrightarrow kk40citizenship$ $dobno \oplus \{personno? \mapsto dobno?\} \in kk01personno \nrightarrow kk40dobno$ $dob \oplus \{personno? \mapsto dob?\} \in kk01personno \nrightarrow kk40dob$ $disablecode \oplus \{personno? \mapsto disablecode?\} \in kk01personno \nrightarrow kk02disablecode$ $taxno \oplus \{personno? \mapsto taxno?\} \in kk01personno \nrightarrow kk01taxno$ $epfstatus \oplus \{personno? \mapsto epfstatus?\} \in kk01personno \nrightarrow kk01epfstatus$ $epfno \oplus \{personno? \mapsto epfno?\} \in kk01personno \nrightarrow kk40epfno$ $socsotype \oplus \{personno? \mapsto socsotype?\} \in kk01personno \nrightarrow kk01socsotype$ $socsono \oplus \{personno? \mapsto socsono?\} \in kk01personno \nrightarrow kk01socsono$

PreUpdateQualification	*personno*? ∈ *staff* *institutionname*? ∈ *kk04institutionname* *graduatedate*? ∈ *kk04graduatedate* *cert*? ∈ *kk04certcode* *course*? ∈ *kk04course* *qualificationn* ⊕ {*personno*? ↦ {*institutionname*?, *graduatedate*?,*cert*?,*course*?} ∈ *kk01personno*? ⇸*degree*
PreUpdateSpouse	*personno*? ∈ *staff* *name*? ∈ *kk40name* *task*? ∈ *kk03taskcode* *employer*? ∈ *kk03employer* *taxstate*? ∈ *kk01state* *spouse* ⊕ {*personno*? ↦ {*name*?, *task*?, *employer*?, *taxstate*?}} ∈ *kk01personno* ⇸ ℙ *Independence*

4.3 Proof of Properties

For a particular specification there may be certain other properties which are desired consequences. These properties may be demanded in the informal requirements for the specification. This section will show a few properties that are not met by the actual system. Below is the list of the properties.

- If a person is not a member of the institution, his personal record will not be in the database. Thus he will not be able to update his personal record.
- If a staff just got married, entering the detail of the spouse should automatically change his marital status from *bachelor* to *married.*
- If a couple, both working at the same institute, any changes made to the employee will update the spouse record.
- If a couple, both working at the same institute, the details of children should be the same regardless of whose record is checked.
- If the staff is a female staff, her record should not be able to accept more than one partner.

For the first item, the following will show that if a person is not a staff, then the operation of *UpdateStaffData* will not be successful. To ensure that the operation of the schema is successful, one of the constraints that has to be satisfied is

personno? ∈ *staff*

However, if the requested staff number, i.e. *personno* is not in the database, which means the following expression:

personno?∉ *staff*

is true. This contradicts the first predicate. Thus, the operation will not be carried out. However, the unsuccessful operation will be carefully handled by error handling schema. In this case, the schema *NotFoundErr* will handle the above problem. This schema will successfully report that the requested staff number is not available and thus generate an error message. Since an error message is generated, it shows that this particular property is fully captured.

For the second item, the following will be shown:

UpdateSpouse | *Person.maritalstatus* = *bachelor* ⊢ *Person.maritalstatus* = *married*

That is, given the declaration of the *UpdateSpouse* operation, together with the requirement that the person is a bachelor, any updates that concern the partner record will change his marital status from bachelor to married. The schema references may be expanded to give the next level of detail as shown below:

≡Person
ΔAllIndependence
personno? : kk01personno
name? : kk02name
task? : kk03taskcode
employer? : kk03employer
taxno? : kk03taxno
taxstate? : kk01state
spousedetail! : Independence |
person? ∈ *staff*
spousedetail!.name = name?
spousedetail!.taskcode = task?
spousedetail!.employer = employer?
spousedetail!.taxno = taxno?
spousedetail!.taxstate = taxstate?
spouse' = spouse ⊕ {person? ↦{spousedetail}
Person.maritalstatus = bachelor
⊢
Person.maritalstatus = married

However, the conclusion,

$$Person.maritalstatus = married$$

which means there is an update in the schema *Person* contradicts with the declaration of $\equiv$ *Person* which says that the property of *Person* should not be changed. Therefore, it is shown that this part of the system properties is not captured.

For the third condition, the domain of the function *spouse(x)* which means that a spouse to the employee *x* will be stored as type *Independence* and since there is no relation between *Independence* and *kk01personno,* then the spouse of *x* will be treated as another outsider even though the spouse is an element of *staff.*

The fourth condition follows from the third one. Since there is no clear relationship to show a possibility of having a spouse to be an employee, i.e. *kk01personno*, any record such as the details of children related to the employee will be treated as his or hers. If a spouse is another employee, then again any records related to him or her will be treated as his or hers, even though, by right they should share the same record. Thus any update to be made to the details of the children will need to be duplicated to the spouse children's record.

The theorem of the fifth condition is:

$$\exists\ x : kk01personno \mid sex(x) = female$$
$$\vdash \#spouse(x) \leq 1$$

Since spouse is defined to be a function from *kk01personno* to $\mathbb{P}$ *Independence*, i.e. $spouse: kk01personno \rightarrow\!\!\!\!\!+ \mathbb{P}$ *Independence,* the range of the function will always have a size of at least zero. Thus, *spouse(x)* may have more than 1. Regardless of the sex of the employee, the number of partner that an employee may have will always be more than one.

5.0 CONCLUSION

Formal specification technique has been successfully applied to a module of an existing system - the human resource information system. By using initial state theorems and preconditions, it has been shown that the system in general, is consistent. However, some errors have been found These errors can be summarized as follows:

- Female employee record is allowed to record more than one husband where it should not be.
- Data for a husband and wife are treated differently. It means that if the wife has given birth to another child, updating the wife record will not affect the husband's record.
- Updating an employee's spouse record will not change an employee marital status. For example if an employee has just married, updating his spouse's record from nil to a new set of data will not affect his/her marital status. This happens because there is no bridge to link a partner record with an employee record.

The above analysis has shown that the formal methods are able to find errors that have not been identified through testing. It is interesting to note that these errors have not been identified even though the system has been running for quite some time.

REFERENCES

[1] H. K. Stephen, *Metrics and Models in Software Quality*, Addison-Wesley, 1995.

[2] A. H. Jeffrey, F.G. Joey, S.V. Joseph*, Modern Systems Analysis and Design*, Benjamin Cummings, 1996

[3] B. K. Boehm, *Software Engineering: R&D trends and defence needs*, M.I.T Press, 1979.

[4] S. P. Roger, *Software Engineering, A Practitioner's Approach*, 4th Edition, Mc-Graw Hill, 1997.

[5] P. Ben, S. Jane, T. David, *An Introduction to Formal Specification and Z*, 2nd Edition, Hemel Hempstead, 1996.

[6] A. Mohd. Zin, R. Maskuri, "Z Specification of A Human Resource Information Systems", Technical Report Series, Fakulti Teknologi dan Sains Maklumat, UKM, July 1998.

[7] J. C. P Woodcock, "Calculating Properties of Z Specifications", ACM SIGSOFT*, Software Engineering Notes*, Vol. 14, No. 5, July 1989.

BIOGRAPHY

Rohidah Maskuri received her BS degree in Computer Science from San Jose State University, California, USA in 1993. She is currently attached to the Information Technology Division, Sepang Institute of Technology. Her research interests include Formal Methods and Programming Languages.

Abdullah Mohd. Zin received his Ph.D. from the University of Nottingham, United Kingdom in 1993. He is currently attached to the Faculty of Technology and Information Science, University Kebangsaan Malaysia.