The Simple Oven

(Essay on COSMIC functional convergence)

Overview of the case study

This case study presents in parallel the specification of the piece of software of a small oven and its COSMIC measurement. This was considered an interesting focus because both approaches are intended to be concerned with the functionality of the piece of software. The functional specification ensures that the functionality is rigorously defined while the COSMIC measurement ensures that the quantity of functionality of the same piece of software is measured.

Despite not using the same language (specification language and COSMIC Model) to describe the same functionality, the specification and COSMIC measure show similarities in many aspects. This similarity includes state variables, and functional structure for the most immediate features.

Another motivation for doing the study is the question of the place of the COSMIC measurer. Should he/she be in the development team or external to it? It would seem that there wouldn't be any obstacle to both specialists working together. And this pairing would be beneficial to the process due to the combination of both perspectives.

The dual presentation of specification and measure is based on the simple oven of Nagano. The choice of the small oven was made because this application was simple and clearly described in natural language. The consequence of such simplicity would be that the principles of this demonstration wouldn't be obscured by the technical complexities of a larger application, and thus hopefully, enabling the emergence of further interesting topics.

The chosen method of specification is the Z method. Z is a proven method for functional specification, and part of the curriculum of "computer sciences" since the 70's, and it became an ISO method in 2002. This paper asserts that the functionality of the small oven is correctly described, then referenced, by its Z specification. The result is that the COSMIC measure is unambiguously related to the specified functionality. This may ensure a better comparison of both specification and measurement approaches.

Requirements for a simple oven

The case study was for a simple oven.

The objective is cooking, and the simple oven has a door and a start button. The functions are cooking, interrupting of cooking, and lengthening of cooking period.

Cooking starts with pressing the button and ends one minute later. Opening the door during cooking suspends cooking. Pressing the button during cooking lengthens cooking time. The same button is used for lengthening the cooking time and starting the oven.

While cooking or while the door is open, the oven light is on.

From: Shin'ichi, NAGANO,Tsuneo, AJISAKA, Department of Design and Information Science, Wakayama University, Japan [4]

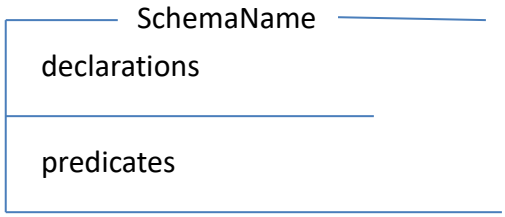
Notes:

1. The initialization of the oven is out of the scope of this case study. Necessary parameter values are already loaded in the data store of the software component.
2. The simple oven is initially in its 'standby' state. This is characterised by the simple oven being 'powered on', initialized, with the door close, and light off.
3. All modelling concerns solely the behaviour of the embedded software component of the simple oven as opposed to the behaviour of the simple oven as a physical device.
4. Pressing the button during cooking lengthens the cooking time by one minute.
5. When the cooking time has elapsed the simple oven returns to its 'standby' state.

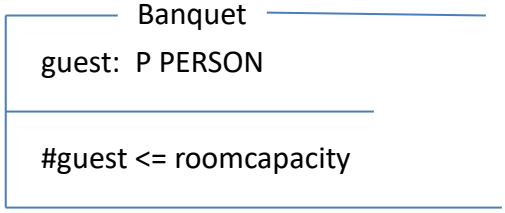
Overview of the Methods Employed

Z-based Specification

Z is a special specification language created by Jean-Raymond Abrial of France in the 70's and further developed by a team of the Programming Research Group of Oxford University in England under the leadership of Professor C.A.R. Hoare [5]. The foundation of Z resides in the description of software functionality by means of set theory and predicate calculus. This Z description of 'what' the software is expected to be doing, as opposed to 'how' it is doing it, is its functional specification. Any specification written in Z is mathematically correct, complete hence error free. A 'formal specification' is an exact description of the functionality of a piece of software. This is the method of choice over natural language written or illustrative 'specification'. This is so much so that Z and its variants are part of standard practices in automation and safety critical software.
The Z notation is applicable to any software application from small to large. However, for large applications the quantity of logical artefacts may quickly become unmanageable therefore the schema, which is so particular to the Z notation, has been devised.
The schema used in Z is a named double box drawn as the pictorial description shown on the right. The name "SchemaName" identifies the schema. The top box contains lines of declarations. This is where the variables are declared. The bottom box contains the list of predicates which must prescribe the logical relationships and constraints that are stated for the variables.



As a simple example, the schema "Banquet" describes the state of the guests to the banquet. The variable 'guest' declares the guest as a collection of persons. The bottom box specifies that the number of guests (#guest) shall be lower or equal to the capacity of the room banquet (roomcapacity).



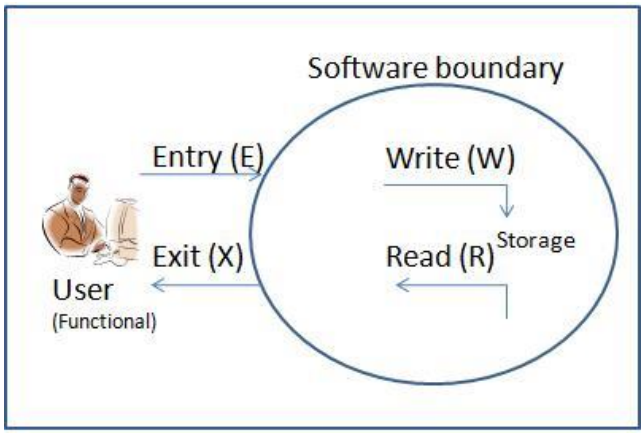
A schema may contain the specification of an amount of functionality from small to large depending on the function to be functionally described. The specification of the functionality of a software application may necessitate a number of schemas. Thanks to some other features available in schema, and not represented in these simple examples, the schemas of a software application will be logically related thus enabling the logical proving of the software application.

Therefore the functionality of a piece of software can be described by means of the schemas of the Z specification method. Z is ISO standardized in ISO/IEC 13568:2002.

COSMIC and Functional Measurement

We are proposing here a simple introduction to the standard COSMIC of functional measurement to help the reading of this essay. But, the measurement situation in an industrial or commercial context requires the measurer to be trained in COSMIC or carries out a thorough reading of the measurement manual listed in [2]. In functional measurement we focus on the functionality embedded in a piece of software. Also, 'Agile' or not, the functionality is the main feature of the software used by the end-user, purchased by the customer, contracted on and insured in outsourcing the contract. This functionality is what is measured by COSMIC.

The functionality is the service rendered to the user in terms of the information he can enter (Entry), the information he can safeguard (Write), retrieve (Read), or obtain from the software (Exit). Each of these activities is called a Data Movement because they move a group of data representing the user's required information. The standard prescribes a set of rules to respect while listing these Data Movements.



The piece of software to be measured is enclosed in a boundary. The functional user is external to the boundary. All exchanges Entry or Exit are valid only when they cross the boundary. Any Write or Read is valid only when they make data persistent in a store or retrieve persistent data from a store.

The required functionality is described in Functional User Requirements (FUR). The FUR can be analysed into a number of Functional Processes, each representing a piece of functionality that makes sense to the Functional User. Once triggered by a Triggering Event the behaviour of each Functional Process is described by a collection of Data Movements Entry, Write, Read, and Exit.

The unit of functional size is the CFP (COSMIC Function Point) which is one Data Movement.

The arithmetic sum of Data Movements is the functional size of the piece of software.

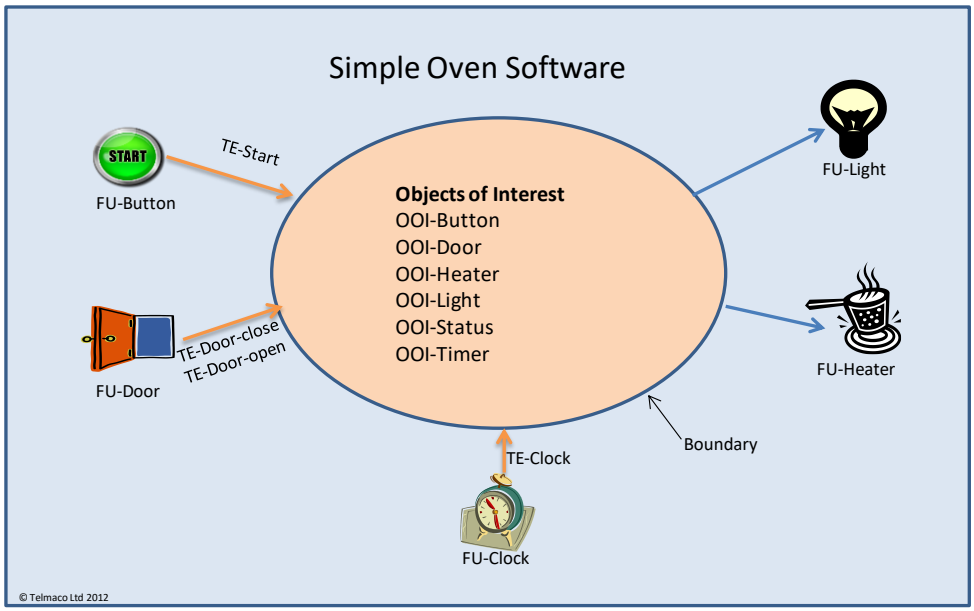
Context Diagram of the Simple Oven Software

The functionality of the Simple Oven software is represented within the oval shape and the border line of this shape is its functional boundary. The COSMIC Functional Users (Logical artefacts' name are prefixed 'FU') are shown around the shape. In COSMIC terms we have:

- FU-Light - a recipient of the command to the light to be set on or off
- FU-Heater – a recipient of the command to be set on or off
- FU-Clock - a sender, every second, of a timing information when it senses the Triggering Event TE-Clock
- FU-Door – a sender of the information door-open and door-close when it

respectively senses the Triggering Events TE-Door-open and TE-Door-close
- FU-Button – a one-shot button which sends the information button-pushed when it senses the Triggering Events TE-Start.

Inside the boundary are located the (logical) Object of Interest (prefixed OOI). These are the artefacts from the world of the Functional Users about which the Simple Oven software is required to process and/or store data.



The recognised OOIs and their Data Groups (DG) are then:
- OOI-Door – Identification of door open and close;
  - DG-Door (2 attributes)
- OOI-Button – Identification of the button start;
  - DG-Button (2 attributes)
- OOI-Light – Commands Light-ON and Light-OFF;
  - DG-Light (2 attributes) (*)
- OOI-Heater – Commands Heater-ON and Heater-OFF;
  - DG-Heater (2 attributes) (*)
- OOI-Timer – Identification of the clock tick and the timer data to be processed;
  - DG-Clock (1 attribute)
  - DG-Timer (1 attribute) (*)
- OOI-Status – 4 distinct status information and the current one to be processed.
  - DG-Status (4 attributes) (*)
  - DG-Status-Current (1 attributes) (*)

(*) - These attributes have been made persistent at initialisation and stored in the COSMIC "Persistent Storage" of the Simple Oven software. It was thought convenient to extend the context diagram to include the Persistent Storage of the OOIs since these shall logically reside within the boundary of the functionality.

The Simple Oven as a Finite State Machine

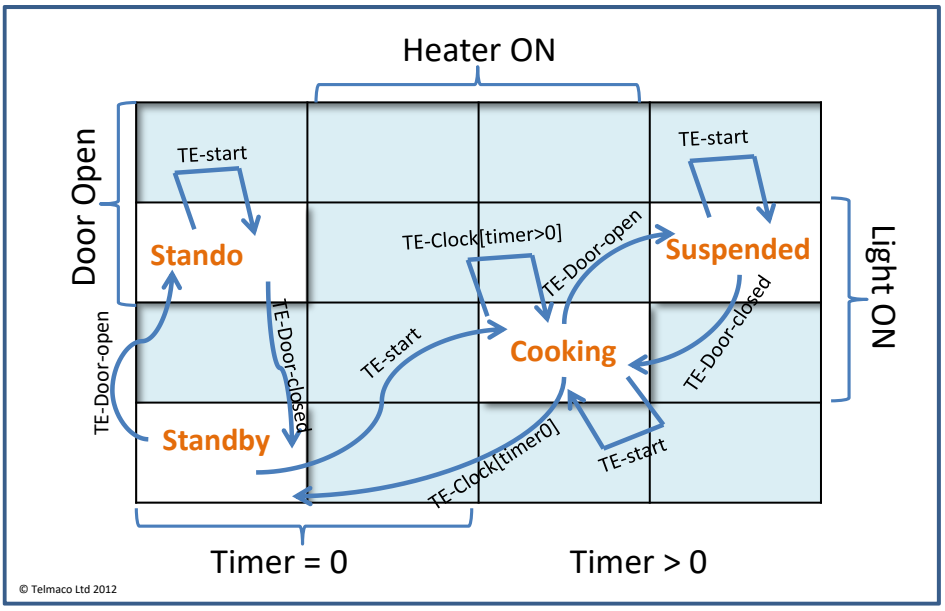
Karnaugh Diagram of the Requirements

The Karnaugh diagram is issued, in 1953, from the work of Maurice Karnaugh when

applying Veitch diagram to logical circuitry. It revealed itself as a useful method to simplify Boolean algebra expressions. But it is also useful to represent the states of Finite State Machines. Hence the 4 X 4 diagram below.
Four COSMIC Functional Users are:

- FU-Door is represented by its values DOOR Open and its opposite Door Close;
- FU-Heater is represented by its values Heater ON and its opposite Heater OFF;
- FU-Light is represented by its values Light ON and its opposite Light OFF;
- FU-Clock is represented by the Timer value equals to 0 or larger than 0.
- FU-Button – see below.

Note: The 'one-shot' button is not necessary for defining the states but it is useful for enacting the transitions between states through the Triggering Event (TE-start) it effects. Therefore it does not show on the Karnaugh diagram. However the button behaves like a Functional User as a sender of triggering information. Hence its functional behaviour needs to be taken into account in the specification and the COSMIC measurement.



The four required states are shown, namely:

- State "Standby" is also the 'initial state' resulting from (i) the initialisation process not shown here and (ii) from the end of cooking;
- State "Stando" is the state acquired from "Standby" when opening the door;
- State "Cooking" is the active state obtained when pressing the one-shot button Start;
- State "Suspended" is obtained by opening the door during the cooking, hence the cooking is temporally stopped and the clock de-activated.

The four COSMIC Triggering Events are shown in action ensuring the required transitions between the states:

- TE-Clock sensed by FU-Clock which triggers the decrease of the Timer every 1 second when the status is "Cooking" or triggers a transition to "Standby" when the timer reaches zero;
- TE-Door-open is sensed by FU-Door which triggers a transition to "Stando" or "Suspended";
- TE-Door-close is sensed by FU-Door which triggers a transition to "Standby" or to

"Cooking";

- TE-Start is sensed by FU-Button which triggers a transition to "Cooking" or which maintains the current state, or lengthens cooking time.

Z representation of the States of the Small Oven

The 'state' of the simple oven is given by a set of variables, the state variables, which are conditioning the status of the simple oven. The state variables are defined as a 'free types' with their values enumerated: DOOR, HEATER, and LIGHT. The state variable TIMER is the set of the natural numbers starting with 0.

The type STATUS has been added in order to have a denomination incorporated with the state variables. At any one specific time, the simple oven can only be in one of the four states: standby, stando, cooking, or suspended. This characterises the 'invariant' of the functionality.

Even though it is not part of the State the start button plays a determining role in triggering the cooking activity. It is a non-functional requirement that the start button is a one-shot button. Then, once "pressed" it will be immediately "released".

section *Document2* parents *standard_toolkit*

$BUTTON ::= pressed \mid released$
$DOOR ::= open \mid close$
$HEATER ::= heaterON \mid heaterOFF$
$LIGHT ::= lightON \mid lightOFF$
$STATUS ::= standby \mid stando \mid cooking \mid suspended$
$TIMER == \mathbb{N}$

The State

The general state of the simple oven is described by showing how the main parameters interrelate at rest, during cooking, and with door open. The state of the simple oven at any point in time is a 4-tuple drawn from:

Oven-State == DOOR × LIGHT × HEATER × TIMER

The schema "SimpleOven" specifies the state of the simple oven showing the declarations in the upper box and the 'invariant' aspects of the state in the bottom box.

SimpleOven

$door : DOOR$
$light : LIGHT$
$heater : HEATER$
$stt : STATUS$
$tm : TIMER$

$stt = \text{standby} \Rightarrow door = close \land light = \text{lightOFF} \land heater = \text{heaterOFF} \land tm = 0$
$stt = \text{stando} \Rightarrow door = open \land light = \text{lightON} \land heater = \text{heaterOFF} \land tm = 0$
$stt = cooking \Rightarrow door = close \land light = \text{lightON} \land heater = \text{heaterON} \land tm > 0$
$stt = \text{suspended} \Rightarrow door = open \land light = \text{lightON} \land heater = \text{heaterOFF} \land tm > 0$

One particularly interesting state is the initial state. That is the state of the simple oven when it is just powered on, unused, with the door close. This is also the "Standby" state which is defined by the schema "StandbyState" below. Let's note that the declaration box of the schema does not need to repeat the collection of declarations listed in the schema "SimpleOven" because this list is represented by the simple oven state schema "SimpleOven" in the declaration box.

StandbyState

SimpleOven
stt = standby

Both descriptions of the simple oven as a finite state machine, the first by the Karnaugh diagram then the second by the Z notation assert similar state information about the simple oven. They describe the four simple oven states allowed by the requirements. They also show that the state consideration is static; hence the functionality is not exercised in the description of this initial state. On the other words, there is not yet any functionality. Therefore, there is nothing for COSMIC to measure.

Clocking the Oven until end of cooking

Z-specification of the clocking functionality

The clocking functionality is triggered every second by the tick of the clock. The state variables involved are defined in the top box decorated with ‘ to identify the post-conditions.

The pre-condition of the clocking functionality is for the simple oven to be in the status ‘cooking’. That is, the content of the timer has a reserve of time, the light is on, and the heater is active. Without these conditions being realised the clocking functionality must be inactive; that is the timer is paused.

The post-condition of this functionality prescribes that either the timer has a reserve of time then the timer would be decreased by one second or the timer has reached zero then the status of the simple oven must return to ‘standby’ with the light and the heater turned off.

Therefore all this behaviour is specified in the bottom box of the “Clocking” schema.

Clocking

Δ*SimpleOven* *stt*′: *STATUS* *tm*′: *TIMER*
tm >0 ∧ *stt* = *cooking* ⇒ *tm*′ = *tm* – 1 *tm*′ = 0 ⇒ *stt*′ = *standby*

COSMIC measure of the clocking functionality

The Functional User FU-Clock sends timing information when it senses, every second, the Triggering Event TE-Clock. This starts the Functional Process “FP-Clocking”.

The timing information will be activated only if the status of the simple oven is ‘cooking’. Hence the first action of the simple oven will be to check the status.

Status permitting, the Functional Process will decrease the timer by one second. If the timer has reached zero FP-Clocking will ensure the passing of the simple oven to the ‘standby’ state.

Functional Process: FP-Clocking

Description: The clock decreases the time counter by 1s and process the Timer = 0 transition to status 'standby'

TE-Clock

Data Movement	Type	Entry	Write	Read	Exit
DM-Initiate timer		1			
DM-Retrieve the current status of the small oven				1	
DM-Retrieve the time counter value				1	
DM-Decreases the time counter by 1s and store it			1		
DM-Timer at 0 - retrieve the heater control				1	
DM-Timer at 0 - exit control to stop the heater					1
DM-Timer at 0 - Retrieve the turn off light control				1	
DM-Timer at 0 - Exit control to turn the light off					1
DM-Timer at 0 - Retrieve the code for status 'standby'				1	
DM-Timer at 0 - Register status 'standby' as current			1		
Total Base Size (CFP)	**10**	**1**	**2**	**5**	**2**

Pushing the button "Start"

Z-specification of the cooking functionality

The cooking is triggered by pressing the button 'Start' when the software is in the 'standby' state. This is the pre-condition. Then, follows the post-condition with the simple oven being in the 'cooking' state with the button being released.
But 2 other pre-conditions exist. The first is when the initial state is 'stando' and the simple oven remains in the state 'stando'. The second is when the simple oven is in state 'cooking'. In this latter case the state remains 'cooking' but the timer is increased by 60 seconds thus lengthening the cooking time. No other variables are changed.

StartCooking

$\Delta SimpleOven$
$bt : BUTTON$
$light' : LIGHT$
$heater' : HEATER$
$stt' : STATUS$
$tm' : TIMER$
$bt' : BUTTON$

$stt = \text{standby} \land bt = pressed \Rightarrow stt' = cooking \land bt' = released \land tm' = 60$
$stt = \text{stando} \land bt = pressed \Rightarrow stt' = \text{stando} \land bt' = released$
$stt = cooking \land bt = pressed \Rightarrow stt' = cooking \land bt' = released \land tm' = tm + 60$

COSMIC measure of the cooking

The Functional User FU-Button sends its identification when it senses, the Triggering Event TE-Start. This starts the Functional Process "FP-Button-Start".

The first action of the simple oven will be to check the status.
When the simple oven is in the ‘standby’ state, its state is passed to the ‘cooking’ state with the heater and the light turned on. Also, during this transition the timer is initialised to 60 seconds.
Following an initial state ‘stando’ or ‘suspended’ nothing is changed.
From a state ‘cooking’ the timer will be incremented by60 seconds and nothing else will change.

Functional Process: FP-Button-Start

Description: Following an initial state 'stando' or 'suspended' nothing is changed. From a state 'cooking' the timer will be incremented by 60 seconds and nothing else will change

TE-Start

Data Movement	Type	Entry	Write	Read	Exit
DM-Receive the button stimulus		1			
DM-- Retrieve the current status of the small oven				1	
DM-Retrieve the control to set the heater ON				1	
DM-Exit the heater ON control					1
DM-Retrieve the control to set the Light ON				1	
DM-Exit the light ON control					1
DM-Set the timer to 60s			1		
DM-Retrieve the Status code 'cooking'				1	
DM-Set the current status to 'cooking'			1		
DM-Status 'cooking': Retrieve the current timer value				1	
DM-Status 'cooking': Increase timer value and store it thus increasing cooking time			1		
Total Base Size (CFP)	**11**	**1**	**3**	**5**	**2**

Opening the Door

Z-specification of the door opening functionality

Opening the door of the simple oven can happen only from states ‘standby’ and ‘cooking’. In other states the door is already open.
This is specified by the schema “OpentheDoor”. In its declarations the schema takes reference from the state schema “SimpleOven” which specifies the invariants the simple oven must satisfy. The variables affected by the post-conditions are also declared with door’, light’, and stt’ for the state of the simple oven.
The bottom box of the schema shows the predicates controlling the opening of the door from the states ‘standby’ and ‘cooking’.

OpentheDoor

$\Delta SimpleOven$

$door'$: *DOOR*

$light'$: *LIGHT*

stt': *STATUS*

---

stt = standby ⇒ stt' = *stando*

stt = *cooking* ⇒ stt' = *suspended*

COSMIC measure of the door opening functionality

The Triggering Event TE-Door-open is sensed by FU-Door which then triggers the Functional Process "FP-Opening the door". The first action of the small oven will be to check the status.

When the small oven is in the 'standby' state, its state passes to the 'stando' state with the light turned on.

From the state 'cooking' the small oven is transitioned to state 'suspended' where the heater is turned off, the timer is paused and the light still on.

Functional Process: FP-Opening the door

Description: The Triggering Event TE-Door-open is sensed by FU-Door which then triggers the Functional Process "FP-Opening the door". The first action of the small oven will be to check the status

TE-Door-open

Data Movement	Type	Entry	Write	Read	Exit
DM-Small oven recognises the door being open		1			
DM-Retrieve the oven current status				1	
DM-'standby' status: Retrieve the light ON control				1	
DM-'standby' status: Exit the set light ON control					1
DM-'cooking' status: Retrieve the heater OFF control				1	
DM-'cooking' status: Exit the heater OFF control					1
DM-Retrieve code status 'stando' or 'suspended'				1	
DM-Store the new current status 'stando' or 'suspended'			1		
Total Base Size (CFP)	**8**	**1**	**1**	**4**	**2**

Closing the Door

Z-specification of the closing of the door

Closing the door of the simple oven can happen only from states 'stando' and 'suspended. In other states, the door is already close.

This is specified by the schema "ClosingtheDoor". In its declaration section the schema takes reference from the state schema "SimpleOven" which specifies the invariants the simple oven must satisfy. The variables affected by the post-conditions are also declared with door', light', and stt' for the state of the simple oven.

The bottom box of the schema shows the predicates controlling the closing of the door from the states 'stando' and 'suspended'.

ClosingtheDoor

$\Delta SimpleOven$

$door'$: *DOOR*

$heater'$: *HEATER*

$light'$: *LIGHT*

stt': *STATUS*

---

stt = stando $\wedge$ $door = open \Rightarrow stt' = standby \wedge door' = close$

$stt = suspended \wedge door = open \Rightarrow stt' = cooking \wedge door' = close$

COSMIC measure of closing of the door

The Triggering Event TE-Door-close is sensed by FU-Door which then triggers the Functional Process "FP-Closing the door". The first action of the small oven will be to check the status.

When the small oven is in the 'stando' state, its state passes to the 'standby' state with the light turned off.

From the state 'suspended' the small oven is transitioned to state 'cooking' where the heater is turned on, and the light still on.

Functional Process: FP-Closing the door

Description: The Triggering Event TE-Door-close is sensed by FU-Door which then triggers the Functional Process "FP-Closing the door". The first action of the small oven will be to check the status.

TE-Door-close

Data Movement	Type	Entry	Write	Read	Exit
DM-Recognise the door closing stimulus		1			
DM-Retrieve the current status of the small oven				1	
DM-'stando' status: Retrieve the control for light OFF				1	
DM-'stando' status: Exit the light OFF control					1
DM-suspended' status: Retrieve the control for heater ON				1	
DM-'suspended' status: Exit the heater ON control					1
DM-Retrieve the status code 'standby' or 'cooking'				1	
DM-Store the current status			1		
Total Base Size (CFP)	**8**	**1**	**1**	**4**	**2**

Conclusions

COSMIC Measurement Summary for: Simple Oven Software

The total measured base size of the functionality of the simple Oven software is:

	Size	Type	Entry	Write	Read	Exit
	Base Size (CFP)	**37**	**4**	**7**	**18**	**8**

	Total	37 CFP (v3.0)

Observations for further developments

A number of observations can be reported at this preliminary stage. The overall structures of specification and measurement display a remarkable alignment between Z Schemas and Functional Processes. There is an almost perfect correspondence between Z state variables and COSMIC Objects of Interest. There is then an observational correspondence between the representations of the Z specification and the COSMIC measure. It is uncertain at this stage whether this correspondence is scalable for larger sizes of functionality (i.e. several 100 CFP and beyond) and when measurer and specifier are distinct persons. In principle, if the logic of each Z and COSMIC are respected, they may be scalable. Or, perhaps not, and why?

Also, during the formulation of specification and measure, it was helpful to note that the inherent logic of the Z specification helps in confirming the COSMIC measurement steps. And inversely, the logical integrity of the COSMIC Model enables a first check of the Z specification.

Therefore from these initial observations it would be possible to conclude that there is no obstacle at this early stage to assuming that the COSMIC functional representation might be a fair description of the functionality of the piece of software under study here. But would there be any patterns that would let these descriptions functionally diverge?

References

1 Bowen, Jonathan, “Formal Specification& Documentation Using Z”, International Thomson Computer Press, 1996

2 COSMIC, “The COSMIC Functional Size Measurement Method - Measurement Manual”, Version 3.0.1, 2001

3 Lightfoot, David, “Formal Specification Using Z”, Macmillan Computer Science Series, 1991

4 Shin’ichi, NAGANO, Tsuneo, AJISAKA, “Functional metrics using COSMIC-FFP for object-oriented real-time systems”, Department of Design and Information Science, Wakayama University, Japan, 2011

5 Spivey, J. M., “The Z Notation: A Reference Manual”, Published by Oriel College, Oxford England, 1998

6 The functional measurement was produced with MeterIT-Cosmic (tm) - V. 1.8 the automatic COSMIC measurement tool of Telmaco Ltd. For more information or comments on this product contact Telmaco Ltd at info@telmaco.co.uk

7 The Z specification was produced with Z word version 3.0 the Z editor and syntax checker of Software Forge. For more information on Z word and CZT contact Software Forge on http://sourceforge.net/about

Acknowledgement:

Our thanks to Arlan Lesterhuis of COSMIC-MPC and Hassan Soubra for their insightful reviews of this paper. Thanks also to CS, PF and CB from the UKSMA Group for their valuable comments.

