Computational conceptual modelling and requirements capture: a systems approach.

The Dagstuhl Case Study.

Mike Holcombe, Tony Cowling and Kirill Bogdanov,
(Department of Computer Science,
University of Sheffield, Sheffield, U.K.
{m.holcombe, a.cowling, k.bogdanov @dcs.shef.ac.uk})

Acknowledgements: The X-machine Group (Sarah Chambers, Matt Fairtlough, Camilla Jordan, Tony Simons and the 4th year students (*Sheffield*), Marian Gheorghe, Tudor Baleanescu, Florentin Ipate, Horia Georgescu, Christina Vertan (*Bucharest*), Evi Kapeti, George Eleftherakis (*Thessaloniki*))

Abstract We consider the lighting system requirements document and develop a way to map the informal text-based requirements onto a *distributed computational conceptual model*. This apprach is methodical and involves the building of a computational model from which very powerful test sets can be easily generated. This approach highlights the need for considering the test strategy at an early stage of the system modelling.

This paper should be read in conjunction with the original description of the Dagstuhl case study at:

http://www.iese.fhg.de/Dagstuhl/casestudy.pdf

together with the appendix to this paper which will be found at:

http://www.dcs.shef.ac.uk/~wmlh/paper/DagAppend.pdf

Key words and categories:
Computational model, conceptal model, change risk, X-machine, testability.

0. Introduction to computational conceptual modelling.

Conceptual modelling is fundamental to the process of requirements

engineering, since the requirements for a system can only be understood properly if one has an adequate conceptual model of the environment within which that system must operate. For an information system, the conceptual model needs to describe three main aspects:

- the structure of the information that the system is to process;
- the structure of the processing that the system is to perform; and
- the dynamic behaviour of the system as it operates.

0.1 Traditional approaches to conceptual modelling.

In some older analysis and design methodologies, such as SSADM, these three aspects were described as separate models, known respectively as the data, processing and time models. By contrast, more modern methodologies have usually taken a simpler view, in which they just identify the data and processing aspects as separate models. They then divide each of these into a static and a dynamic component, so that what would previously have been described as the time model is regarded instead as the dynamic components of the data and processing models. Even this division into two models is misleading in some respects, though, for in practice the two are so closely inter-related that it would be better to treat them as two components of one model, rather than as two supposedly separate models.

From the perspective of this paper, a more important feature of these methodologies is that all of the common ones have adopted the approach of using diagrammatic structures to represent each of these components of the conceptual model. In some of the early approaches the forms of these diagrams were very loosely defined, so that one could not even identify a precise syntax for the icons that were used in them, or the relationships between the elements of a diagram. More modern approaches have tightened up on the syntax, by adopting notations such as UML, but even here there are currently still problems with defining the semantics of at least some of the diagrammatic structures. This may not be too much of a handicap if one considers that the only purpose for a conceptual model is to aid communication with non-specialists, since it is unlikely that they would be concerned with the finer detail of the semantics. We would want to argue, however, that this is a grossly inadequate view of the role of a conceptual model.

Specifically, the starting point for the approach described in this paper is the belief that a conceptual model of an information system should have a vital role to play in the validation of all subsequent stages in the development of that system. That is, the conceptual model should not just be there to describe the context for the statement of requirements, impor-

tant though that is. Rather, it should also provide the basis for validating any specification of the system that is built, and then for validating the designs for the system, and finally for identifying the test cases that will be used in validating the eventual implementation. In order to play this wider role, though, a conceptual model needs to be more precise than is currently achievable using diagrammatic methods. Hence, the approach that is being investigated in this paper is that of attempting to construct a formal conceptual model of the system that is to be studied, and using the development of this formal model as a guide both to defining those requirements that are clearly fixed, and to identifying areas where the requirements can not yet be determined completely. For a number of important reasons, not least of which is the need to provide a rigorous framework to reason about validation, the conceptual model is based on a full computational model. The Turing model is traditionally the one used for analysis such as complexity and computability theory. This is, however very inconvenient and we choose instead a practical but general model based on generalised machines but with a simple and powerful semantics and structure.

The formalism that is used as the basis for this model is that of the X-machine, since this is one of the simplest formalisms that actually captures all three of the aspects that are required for a conceptual model. Like a finite-state machine, an X-machine has a set of control states, and transitions between them which are triggered by inputs, and which cause outputs to be generated. These features therefore capture the dynamic behaviour of the processing within a system. In addition, though, an X-machine has a memory, and each state transition is associated with a processing function that can act on the memory: to be precise, it is these processing functions which actually use the inputs and the memory values in order to generate the outputs, and possibly update the memory. The structure of the memory therefore captures the static structure of the data, while the processing functions describe how the data changes, and so capture its dynamic behaviour.

Besides its simplicity and adequacy, another advantage of the X-machine model is that there is now a substantial body of results for it concerned with how X-machine models can be tested, and the amount of testing that needs to be done in order to test completely an X-machine which has been constructed from a set of processing functions that have already been tested separately. While the current results are concerned with systems that can be modelled as a single X-machine, or using various refinements of a single machine, work is now in progress on extending these to systems composed of a network of communicating X-

machines. This extension is particularly relevant to a system such as the one in this case study, where there is a natural distribution of the processing that one would expect to see reflected in a conceptual model, even if it were not going to matched exactly by whatever architecture might eventually be designed for an actual system.

Thus, the basis of the approach that is used here is to identify the natural set of communicating components, each of which should be represented separately in the formal conceptual model, and then to develop an X-machine model for each of these components. This will involve trying to extract for each component information that can be used to identify its control state, its inputs and outputs, its memory structure and the processing that it will need to perform on the data in its memory.

0.2. The fundamental questions posed during conceptual compuational modelling.
These relate to the following:

- what is prompting change to the system?
- what prior knowledge does the change depend on?
- what are the observable results that are caused by the system?
- what are the implications for the internal knowledge of the system - in other words how does the prior knowledge get updated, if at all?
- what are the constraints on the requirement?
- is the requirement likely to change?

These questions are asked for each requirement. In some cases a requirement is really a definition of a term and will become part of the system glossary, however, many of the functional requirements will involve all of the above questions. Generally, memory values are not deleted or changed, new values are appended to the existing records. This can be refined later when the implications of overwriting data is clear. The type of information in the memory needed for a process is recorded under the column headed: Memory and the updated memory following a process: Memory'. The results from a process are going to be referred to as Outputs, these are displays, actuator commands or other products from the process.

0.3 Change risk estimation.

The issue of change risk is introduced here. It is likely that the analyst will only be able to guess these values, they should be inserted, however, so that the issues can be explored further with the client(s). Areas of high risk might be identified and the overall design architecture developed to accommodate for this by removing these aspects from central roles in the structure to reduce the problems associated with change and *requirements creep*. In the following the values are purely intuitive and the only general principle applied is that the vaguer the requirement the higher the change risk, since more information is required and the assumptions made by the analyst might not match those of the client. However we can base the values on some general criteria and these are described below.

A suggested set of criteria for identifying the risk levels.

1. A requirement which is fundamental to the nature of the system, and for which the detailed information given is clearly correct. This gives the lowest level of risk.

2. A requirement which is fundamental to the nature of the system, but for which some detailed information is either not given, or is given in a form which suggests that some minor changes might occur.

3. Either, a requirement which is fundamental to the nature of the system, but where the information given is sufficiently loose as to suggest that major changes might occur; or, a requirement which is less fundamental to the nature of the system, but which is apparently documented in sufficient detail that it seems likely to be correct.

4. A requirement which is less fundamental to the nature of the system, but for which detailed information is either not given, or is given in a form which suggests that changes could occur.

5. A requirement which is not fundamental to the nature of the system, and for which there is little information given, or for which there is an explicit indication that change is expected. This is the highest level of risk.

The experience and attitude of the requirements engineer have to be relied on to a certain extent. However, it is better to be cautious and assume that change is likely in the absence of definite information to the

contrary. Paraphrasing Tom Gilb: *if it is not stated to be a fixed requirement then it must be a changeable one.*

The first stage is the initial requirements table, here we identify *Features* as business processes and glossary definitions.

Notation: + means “and”; - means “no change” or not applicable; “(?)” identifies areas for further clarification (highlighted in the change risk column); Ref. is the reference in the informal requirements document; and change risk is initially estimated on the 1(low) to 5(high) scale desribed above.

Dagstuhl Intelligent Lighting System (DILS).

1. Initial requirements analysis.

1.1. General users requirements.
These are summarised in table 1.

Table 1: General users needs.

Feature	Input	Memory	Output	Memory’	Constraint	Ref.	change risk
1.1	-	light scene settings	-	-	glossary	U1	4
1.2	occupier detection fires	light scene settings	turn on light scene	-	-	U2	3?
1.3	occupier detection fires + time delay not exceeded	time delay value (T1)	turn on last light scene	-	-	U3	2
1.4	occupier detection fires + time out	time delay value (T1)	turn on standard scene	-	-	U4	2

Table 1: General users needs.

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	change risk
1.5	wall switch on/off	-	switch light on/off	-	switches for ceiling/window	U5	1
1.6	*light scene parameters set*	-	*confirmation message*	*light scene parameters stored*	-	U6	2
1.6(i)	ambient parameters set	-	confirmation message	ambient parameters stored	-	U7	3
1.6(ii)	default (standard?) parameters set	-	confirmation message	default parameters stored	not control panel (?)	U8	2?
1.6(iii)	default ambient set	-	confirmation message	default ambient stored	not control panel (?)	U9	2?
1.7	time delay value set	-	confirmation message	time delay value stored	not control panel (?)	U10	2?
1.8	outdoor sensor failure or motion sensor failure	-	confirmation message	-	-	U11	3

There are a number of (?) entries in table 1. These are caused by negative and unclear information. At this stage it is not clear what these mean or where some of these activities take place.
Also, we could have separated out the features 1.6 (i) - (iii) into a subsidiary table (say Table 1.6) derived from feature 1.6.

1.2. User needs - offices and other specific types of room.
The next three tables describe features that apply only to named areas of the system, in this case the individual offices numbered 1...N, the other rooms and hallways. As with table 1 it would be possible to separate out the groups of entries for features 2.3 (i) - (iii) into a separate table This would be a useful strategy for a more complex example..
.

Table 2: User needs: offices

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	change risk
2.1	-	light scene = ceiling level + task level	-	-	glossary	U12	4
2.2	-	-	-	-	movable control (?)	U13	2?

Table 2: User needs: offices

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	change risk
2.3(i)	task light switched on/off	light scene	turn task light on/off	-	control panel	U14(i)	2
2.3(ii)	ceiling light switched on/off/ ambient	light scene	turn light scene on(?)/ off/ambient	-	on = last or standard?	U14(ii)	3?
2.3(iii)	ambient setting	-	display confirmation message	ambient setting	-	U14(iii)	2

Table 3: User needs: other rooms

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	Change risk
3.1	-	-	-	-	control panel near door to hallway	U15	2
3.2(i)	-	-	-	-	ceiling light switch on control panel	U16(i)	2
3.2(ii)	ambient setting	-	confirmation message	ambient setting	-	U16(ii)	2

The next user needs are concerned with the hallways.

Table 4: User needs: hallways.

Feature	Input	Memory	Output	Memory'	Constraints	Ref.	Change risk
4.1	occupier detectors fire	safe light scene parameters	safe light scene turns on	-	safety definition	U17	2

Table 4: User needs: hallways.

Feature	Input	Memory	Output	Mem-ory'	Constraints	Ref.	Change risk
4.2	occupier zone entry detectors fire	safe light scene parameters	safe light scene turns on	-	-	U18	2
4.3	switch pressed	-	light change (on/off)	-	-	U19	1

1.3. Facility manager needs.
These are described in table 5.

Table 5: Facility manager needs.

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	Chan ge risk
5.1	daylight sen-sor fires	-	lights switched on/off	-	sensor(?)	FM1	2(?)
5.2	occupier detection fires + timeout	time delay value (T2)	lights switched off	-	applies to hallways only	FM2	2
5.3	occupier detection fires + timeout	time delay value (T3)	lights switched off	-	applies to each room(?) separately	FM3	2(?)
5.4	time delay(T2) value set		confirma-tion mes-sage	time delay(T2) value stored	applies to each hall-way section separately (S1, S2, S3?)	FM4	2(?)
5.5	time delay(T3) value set		confirma-tion mes-sage	time delay(T3) val-uestored	applies to each room sepa-rately(?)	FM5	2(?)

Table 5: Facility manager needs.

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	Change risk
5.6	occupier detection has not fired + FM- off input	-	light switched off	-	any room or hallway - (no-one present?)	FM6	2
5.7	individual-malfunction report	-	malfunction report details	malfunction report details stored	anywhere? anything?	FM7, FM8	4(?)
5.8	energy report request	energy consumption	energy report details	-	all systems?	FM9	4(?)
5.9	malfunction history request	malfunction report details	malfunction history details	-	all systems?	FM 10	4(?)
5.10	manual malfunction entry	old malfunction history	confirmation message	malfunction details recorded	all systems?	FM 11	4(?)

The precise structure of the many reports and system behaviour and performance details are as yet unclear and so are likely to change.

1.4. Fault tolerance requirements.
The final group of requirements are concerned with behavioural patterns

Table 6: Fault tolerance needs.

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	Change risk
6.1	outdoor sensor fails	last outdoor light reading	standard-light scene turns on	failure recorded	ceiling lights on, all rooms and hallways?	NF1 NF2	3(?)

Table 6: Fault tolerance needs.

Feature	Input	Memory	Output	Memory'	Constraint	Ref.	Change risk
6.2	outdoor sensor fails + occupier detector fires	-	lights on	failure recorded	hallways	NF3	2
6.3	motion detector fails	light scene details	light scene on	failure recorded	rooms and hallways	NF4	2
6.4	manual and automatic controls fail	-	lights on	failure recorded	hallways only	NF5	2

that contribute to the quality of the system rather than to its functional requirements. The principal concern is quality of service, including fault tolerance. This is considered in table 6.

2. Architectural and interface constraints.

The purpose of this phase is to determine the hard and soft system boundaries. We start by looking at the relative positions within the overall system/building of the different users and what this means for the organisation of the input and output devices.

2.1 User boundaries.
Each room has a user who is able to interact with the room lighting system at the site of the room. In the absence of any information about restricting user interaction (to, for example, the room's "owner" or other group) it will be assumed that any person in a room can access all the general controls in that room. Thus:

2.1.1. each control panel can be used by any person in the room.
2.1.2. the Facilities Manager has an office which is the only place from where the FM requirements can be accessed (except for maintenance and repair when any part of the system can be accessed by maintenance crew - this might be an additional requirement that is protected in some way, it is not considered further here).

2.1.3. a reasonable principle is that any displays intended for a user/ owner should be visible to that user and to no others except, possibly, the Facilities Manager.

2.2. System boundaries.
In deciding the system and subsystem boundaries we need to take account of the natural architecture of the building as well as the constraints imposed in section 1.

2.2.1 Top level system architecture.
The identification of boundaries leads to a top level architecture of the form illustrated in figure 2.1. here there are a number of processes that require the input into a room's system of information from outside that room system, in particular the information from the outdoor sensors, for example ols4, ols 6. These input sources are indicated separately on the figure 2.1.

Since the objective here is conceptual modelling the communication channels from the room systems to the Facilities Manager can be represented as the a rather simplest form of distributed system; effectively a *star* topology). The implementation could also be based around a single computer or by grouping several rooms together under a single computer control. We abstract away these concenrs and concentrate on the distributed nature of thesystem.

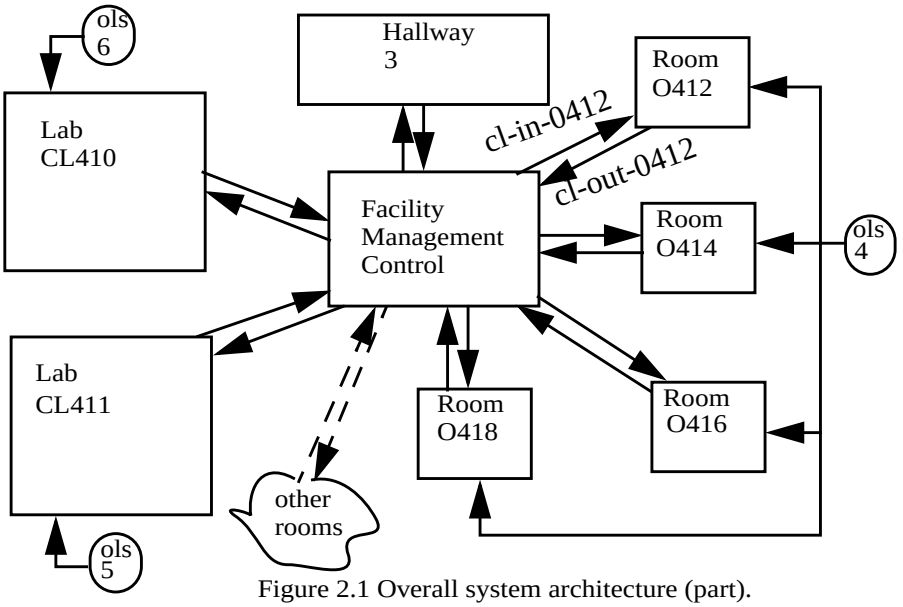


Figure 2.1 Overall system architecture (part).

It should be noted that figure 2.1 only displays a part of the complete system, just enough to demonstrate how it might be described in practice.

2.2.2. Subsystems.
Each individual system will now be decomposed using the process descriptions into a separate system architecture diagrams.

We start with the basic room machine, eg. for O41, and this is illustrated in figure 2.2.

For each input line and output line we should specify the type intended. Thus Boolean would represent {0, 1}. We have to consider also, some other issues relating to how we would model failure states and how the system would detect them. The specific technology used for the implementation would play an important role here.

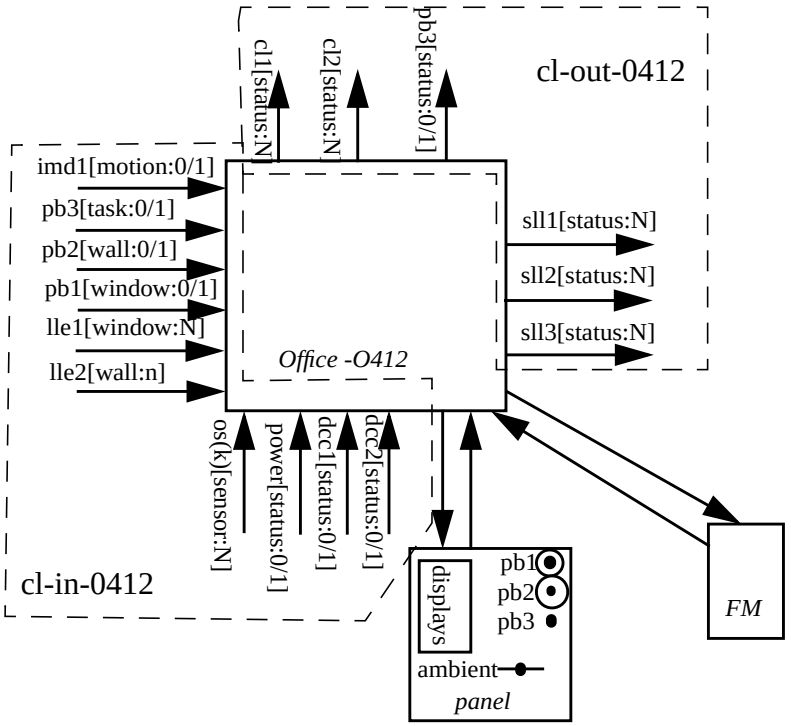


Figure 2.2 Office 0412 system diagram.

Explanation of notation.
An input **xyz[type_name:value1/value2/...]** means that **xyz** is the reference number eg. imd1 (motion detector), **type_name** is the reference to the name of the data involved, say motion detected or not, and the type value is either described abstractly (eg. N, meaning number) or listed explicitly in the case of a finite type such as *Boolean* or its variants. A similar notation is used for outputs.

Such a diagram can be constructed for each subsystem, for instance.
the FM subsystem will be defined in a similar diagram, an outline of which is figure 2.3.

Figure 2.3. Facilities manager subsystem.

Once the individual subsysmore detail. This can be done using a separate communications architecture diagram such as figure 3.1.

3. The communications model.

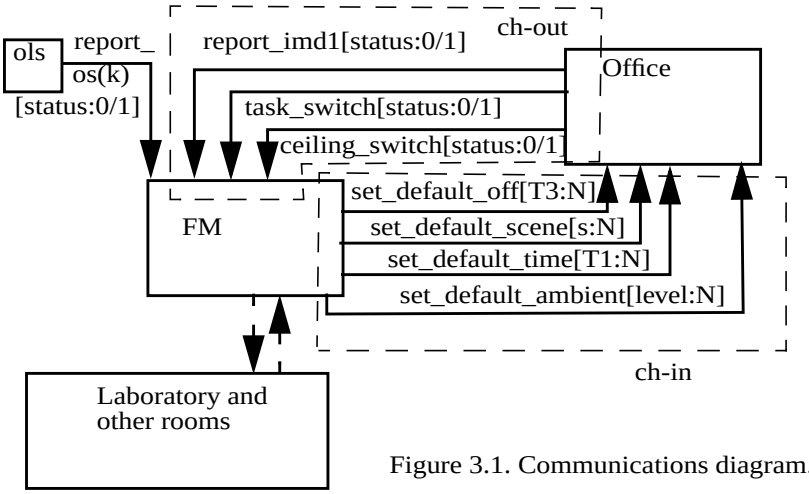


Figure 3.1. Communications diagram.

4. Stream X-machine specifications.

The information presented above will now be turned into a series of stream X-machines. We will be trying to identify the natural subsystem states and the local system memory systems and bringing this information together into a simple and concise description. The subsystem states need to be classified as either *processing* states or *communicating*

states.

The process of machine construction proceeds as follows:

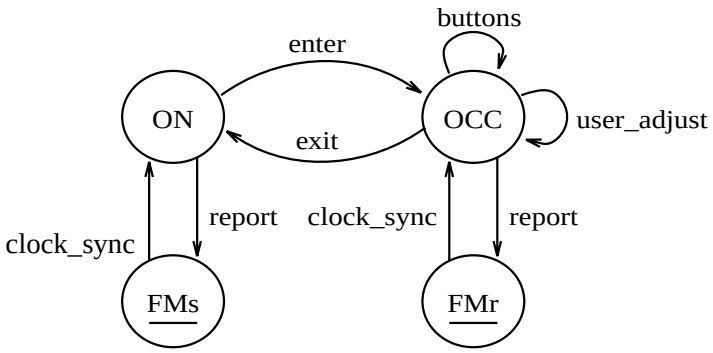
1. Individual processes are formalised as far as possible, their inputs, memory, memory changes and outputs identified and the process is labelled.
2. For each process we decide whether the process must be available either:
    a) always;
    b) almost always (ie. except for certain exceptional situations);
    c) only under certain circumstances - after some other event or process has completed, after a certain period of time etc..
3. States are introduced in order to organise the processes subject to the above constraints.
4. The global memory for the machine is defined at the appropriate level of abstraction;
5. The process functions are formalised.
6. The machine is refined in two ways:
    a) more concrete details are added;
    b) functionality is amended.

Since we are dealing with a distributed system we have to identify the communicating states as well. This will be done here by extending the state space to allow for communication between the FM machine and the other machines in the system. There are a number of issues that need to be resolved. For example, who has priority when FM wishes to interrogate the state of a light scene when a user is changing it. Suppose the FM changes a default time setting (eg. T2) during the operation of a process that needs to know what the default time value is. In the absence of further information one could take a conservative, user-centred approach by giving the user priority where appropriate and not to interrupt processes already begun if defaults are changed.

4.1 The generalised stream X-machine.

This machine is representative of the basic structure of a machine in the system. It has two main modes of operation, on and occupied, and two communicating states corresponding to each of these. We could enhance

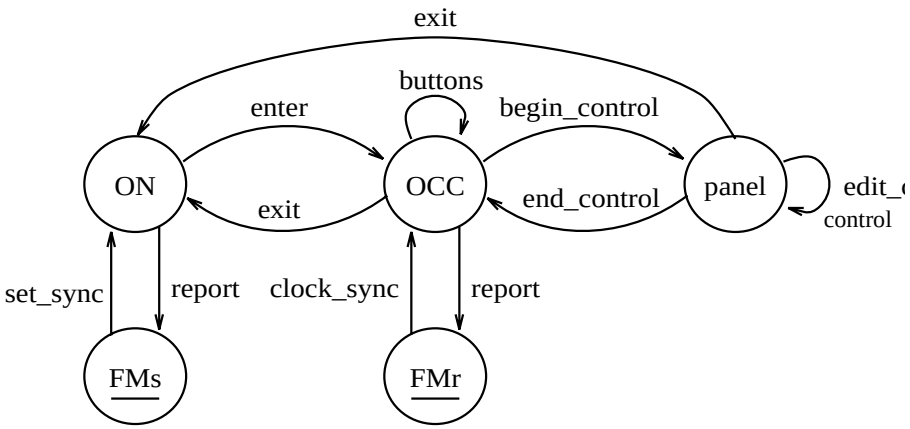
this with other generic features such as an **off** state and a **maintenance** state but we will not dwell on these details.



Original machine

Figure 4.1. GENERALISED MACHINE

4.2 The office stream X-machine.



Refined machine

Figure 4.2. OFFICE MACHINE

In these states users cannot adjust their preferences if a manager tries to change something that might affect what the user is doing. This raises issues of priority that would need to be resolved.

4.2.1 The machine description.

In this section we have not defined the types in the most detailed way for reasons of space limitations.

4.2.1.1 Inputs:- {imd1, pb1, pb2, pb3, os(k), lle1, lle2, dcc1, dcc2, power, PIN}

where imd1, pb1, pb2, pb3, dcc1, dcc2 are **event variables**, os(k), PIN ∈ N,
and lle1, lle2, power ∈ {0,1} are **persistent values**.

4.2.1.2 Outputs:- {cl1, cl2, pb3, sll1, sll2, sll3, power, cl1_status, cl2_status, pb3_status, sll1_status, sll2_status, sll3_status, power_status}

where cl1, cl2, pb3, sll1, sll2, sll3, power, are **commands** and cl1_status, cl2_status, pb3_status, sll1_status, sll2_status, sll3_status, power_status are display **information**,

4.2.1.3 Memory:- light_scene x displays x status x time_defaults x actual time x occupied

where light_scene = name x ambient x desk x list
and list is a list of light sources (l1, l2, l3)
(and so (n, a, d, l) represents a typical member of light_scene with n ∈ name etc).

displays = some suitable set to describe any useful messages or reports [can be defined later represented by δ ∈ displays]
status = cl1_status x cl2_status x pb3_status x sll1_status x sll2_status x sll3_status x power_status - [s = (s1, s2, s3, s4, s5, s6, s7) ∈ status]
time_defaults = T1 x T3 - [(t1, t3) ∈ T1 x T3]]
actual_time = N - [z ∈ N]
occupied = Boolean - [b ∈ B]

In some cases it may be necessary to extend the definition of the memory later when we need to record some information needed by a process. In this method this is done simply by adding on an extra field to the memory type and redefining all previously defined functions to ignore this parameter - *they stay the same*.

4.2.1.4 Basic functions.

Now we define the various process functions, all of which update the room system clock.

enter which detects someone entering the room (imd1 occurs) and reads the memory for the light scene setting, switches the light scene on and refreshes the status (memory) and display [ref 1.2] (dcc1 and dcc2 can help us to decide whether a decision about a room being occupied is reasonable. This aspect would need some further thought as to the physical behaviour of the system.)

buttons: this represents the following five specific functions:

wall_switch is the operation of the wall ceiling switch and the adjustment of the light scene and status by changing the ceiling light and display [ref 1.5]

window_switch is the operation of the wall switch and the adjustment of the light scene and status by changing the window light and display [ref 1.5]

task_light is the operation of the task light switch and the adjustment of the light scene and status and display [ref 2.3(i)]

sw_off is a process that waits for T3 time units and then switches off the lights, it updates the clock afer every time unit.

begin_control a command that starts a user interaction using the control panel (if there was a security issue it would begin with a password or PIN and these would have to be stored in the memory as well as the other information).

edit_control a generic high level function that will be refined later when the precise functionality of the control panel was established. This would include functions to set ambient light levels and to dimm various combinations of lights.

end_control a command signifying the conclusion of the control panel interaction and confirmation of the new settings.

report is the communication request for information from the FM system. It causes the system to enter a communicating state.

clock_sync ends the communication processing if the room is still occupied**.**

set_sync ends the communication processing if the user has left the room in the process of the communication (by referring to the memory value).

exit detects that all occupants have left the room.

The functions can now be defined more formally as illustrated in figure

7, only the first few are documented in detail..

Table 7:

Function	Input	Memory ((n, a, d, l), δ, s, (t1, t3), z, b)	Output	Memory' ((n, a, d, l), δ, s, (t1, t3), z, b)
enter	imd1 =1	((n, a, d, l), δ, 0, (t1, t3), z, 0)	(n, a, d, l) scene turn_on	((n, a, d, l), δ', s', (t1, t3), z+1, 1) where s' is set light scene
wall_ switch	pb2	((n, a, d, l), δ, 0, (t1, t3), z, 1)	display = δ' + wall_light changed	((n, a, d, l), δ', s', (t1, t3), z+1, 1) where s' records new wall light state
begin_ control	PIN	((n, a, d, l), δ, 0, (t1, t3), z, 1)	panel acknowl-edge	((n, a, d, l), δ, 0, (t1, t3), z+1, 1)

It is now straightforward to translate these into precise mathematical functions which describe the memory manipulation and the outputs and control commands.

4.2.2 Design for test conditions.
From the point of view of testing we are now in a position to ask whether the design for test conditions are satisfied. We are going to treat the machine in isolation from the rest of the system and consider the communication requests from the FM as simple basic functions of this machine.

4.2.2.1 Controllability.
Given any particular memory value and a basic function can we always find some input to make the function operate? The answer is no, since if the room is unoccupied we cannot operate the control panel. Hence we must introduce test inputs that will cause the control panel functions, such as **begin_control** to operate under the conditions when the memory has a value with occupied = 0. Call such an input begin_control_test

and adjust the **begin_control** definition to allow for this. We do not have to change the diagram.

4.2.2.2 Observability.
Can we distinguish between any two functions? The key here is in the way the displays are defined. We have not described them in detail but they need to provide enought information to ensure that the basic functions are distinguishable.

4.2.3 The test set constructors.
Here we give a sample. They are expressed as sequences of basic functions. We assume that the implementation has one more state than the specification (k = 1 in the test algorithm), this is purely for illustration. In most cases putting k = 0 is sufficient to generate very powerful test sets in cases like this. There are 35 sequences. For k=1 there are 87 sequences.

4.2.4 The test sets.

The input sequences that trigger the the function sequences are now created using the fundamental test function. This is a recursive procedure that utilises the design for test conditions. Some examples of the test sequences are given at *http://www.dcs.shef.ac.uk/~wmlh/paper/DagAppend.pdf*. As before, there are 35 and 87 sequences.

4.2.5 Refinement.
We mentioned the need to refine the control panel actions (**edit_control**) and these can be done now or later. The test sets can be adjusted automatically once the refinement has been carried out. The refined machine has test sets of 62 (for k=0) and 160 (k=1) test sequences.

4.3 The Facilities Management stream X-machine system.

This is not examined here but involves a similar process to the previous section.

4.4 The complete communicating stream X-machine system.

It would not be sensible to generate the full model and it is unnecessary. The purpose behind the approach is to try to introduce a way of manag-

ing complexity by use of hierarchy and components. Testing the overall system could be carried out using stubs to represent individual machines and just testing the control communications. Smart algorithms to do this are the subject of current research. However, since much of the test generation process is automated it is possible to generate the full test set for the complete model without physically building the complete model. This may involve a substantial computation, however.

Conclusions.
We have described an approach to the construction of a formal specification in a methodical way which we claim is both easy to understand and can be developed in a fairly systematic way.

Novel features include:

- the notation is a convenient blend of diagrams based on finite state machines and simple function definitions;
- the approach has a firm theoretical basis;
- the specification provides a convenient basis for test set generation that can be used in a full implementation;
- design for test issues are highlighted;
- the idea of change risk is introduced;
- an implementation in a standard programming language is straightforward - in fact it can be easily automated.

The introduction of the idea of change risk has helped us to determine some of outlines of the architectural modelling. In a real project, where further discussions with the client are possible, the development of those aspects of the model which are subject to high change risk would be left until later. This would apply, for example to function definitions etc. Another feature of the approach is the methodical way in which the model was developed and concrete details introduced using the structure of the requirements document.

References.

M. Holcombe & F. Ipate. *Correct systems - building a business process solution*, 1998, Springer.

H. Georgescu, C. Vertan and A. Cowling. *A new approach to communicating X-machine systems*, 1999, submitted.

F. Ipate, M. Holcombe et al. *Generating test sequences from non-deterministic X-machines*, 1999, submitted.

T. Balaneascu, A. J. Cowling, H. Gheorgescu, M. Holcombe, M. Gheorghe, C.Vertan, *Communicating X-machine systems are no more than X-machines*. Journal of Universal Computer Science, Volume 5, no. 9, 494-507, 1999.

T. Balanescu & M. Gheorghe, *On testing stream X-machines*, 1999, submitted.

T. Balanescu, M. Gheorghe & H. Georgescu, S*tream X-machines with underlying distributed grammars*, 1999, submitted.

A. Cowling, H. Georgescu and C. Vertan. A structured way to use channels for communication in X-machine systems, 2000, submitted.