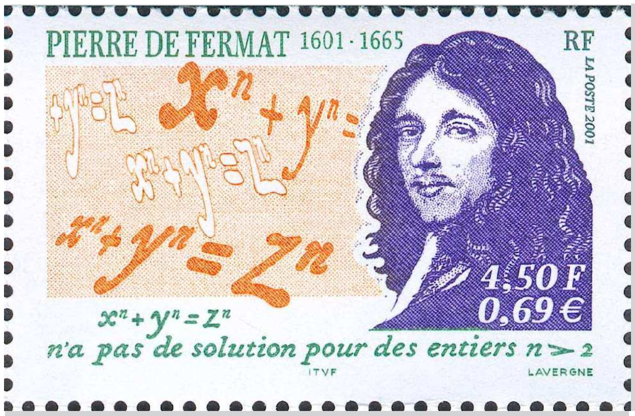

PIERRE DE FERMAT 1601 · 1665
RF
LA POSTE 2001
4,50 F
0,69 €
$x^n + y^n = z^n$
n'a pas de solution pour des entiers $n > 2$
ITVF
LAVERGNE

F E R M A T

Formal Engineering Research using Methods, Abstractions and Transformations

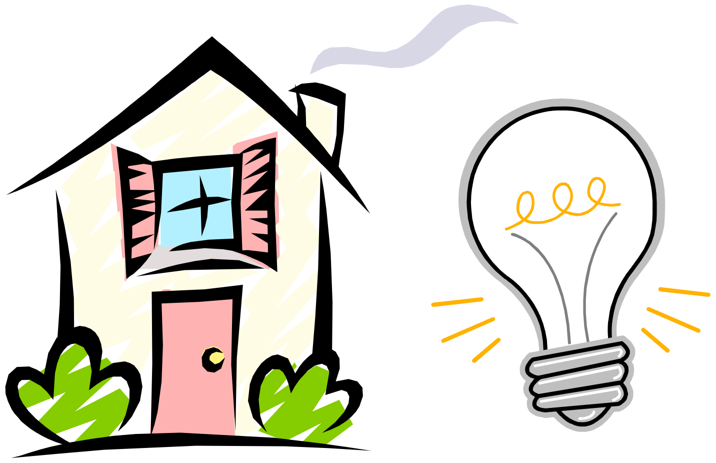
Technical Report No: 2003-11

Extreme Modeling in PROMELA: Formal Modeling and Verification of a Smart Building Control System

Syed M. Suhaib, David Berner
Sandeep K. Shukla

{ssuhaib, shukla}@vt.edu,
david.berner@irisa.fr

Abstract -- In this paper we show the usefulness of an agile formal method (named XFM) based on extreme programming concepts to construct abstract models from a natural language specification of a complex system. In our experience, major challenges faced by industrial formal verification engineers are two fold: (i) Making sure that the natural language specification of a system is translated into a sufficiently complete set of formal properties to be used in model checking of an implementation instance, (ii) In conformance based formal verification using abstraction techniques, creating an abstract model that satisfies all formal properties intended in the natural language specification. Most of the time, it is hard to validate the sufficiency/completeness of the property suite developed from the natural language, or to make sure that the abstract model is constructed correctly. By ``correctly'' we mean that the set of behaviors of the abstract model is not only a superset of the set of behaviors of an implementation, but also a subset (in the best case, equal) to the set of behaviors intended/allowed by the natural language specification. Our XFM based methodology addresses these problems. It is based on building a model in PROMELA along formal linear time properties that are validated with the SPIN model checker. We illustrate the flow on an extensive example of a Smart Building control. It features intelligent control and interaction of illumination, heating, cooling, safety, security, and appliances. We find that this methodology not only constructs abstract models in shorter time than the time taken in constructing ad hoc abstract models from implementation or specification, but also provides models which are constructively correct and closer to the intended specification.



Virginia Tech 1872

Extreme Modeling in PROMELA: Formal Modeling and Verification of a Smart Building Control System

Syed Suhaib
Virginia Tech
Blacksburg, VA USA

David Berner
INRIA/IRISA
Rennes, France

Sandeep Shukla
Virginia Tech
Blacksburg, VA USA

Contents

List of Figures

List of Tables

Abstract

In this paper we show the usefulness of an agile formal method (named XFM) based on extreme programming concepts to construct abstract models from a natural language specification of a complex system. In our experience, major challenges faced by industrial formal verification engineers are two fold: (i) Making sure that the natural language specification of a system is translated into a sufficiently complete set of formal properties to be used in model checking of an implementation instance, (ii) In conformance based formal verification using abstraction techniques, creating an abstract model that satisfies all formal properties intended in the natural language specification. Most of the time, it is hard to validate the sufficiency/completeness of the property suite developed from the natural language, or to make sure that the abstract model is constructed correctly. By "correctly" we mean that the set of behaviors of the abstract model is not only a superset of the set of behaviors of an implementation, but also a subset (in the best case, equal) to the set of behaviors intended/allowed by the natural language specification. Our XFM based methodology addresses these problems. It is based on building a model in PROMELA along formal linear time properties that are validated with the SPIN model checker. We illustrate the flow on an extensive example of a Smart Building control. It features intelligent control and interaction of illumination, heating, cooling, safety, security, and appliances. We find that this methodology not only constructs abstract models in shorter time than the time taken in constructing ad hoc abstract models from implementation or specification, but also provides models which are constructively correct and closer to the intended specification.

1 Introduction

Today's way of building formal models can be compared to what programming was fifteen years ago. From a spoken specification an ad hoc abstract model is built, then formal properties are developed to check if the model satisfies the specification. Figure 1 displays the overall design flow. A formal model checker is used to verify properties in the model. There are several drawbacks in this approach. One drawback is that the ad hoc building of both the model and the properties is error prone and the effort of model building and debugging grows exponentially along with the size of the model. Another foreseen problem is that there is no way of guaranteeing the inclusion of all properties, some of which may be overlooked, thus reducing the significance of the model. Also, if a property fails, it is tedious to debug the model although few indications do exist that tell us about the location of the bugs. Finally, there is a tendency that the model will include more behavior than its specification will allow. Implementation details can get into the abstract model, that make the model have unwanted properties and hence the implementation being checked against it may have these as well. The implementation details in the abstract model may also introduce unwanted complexity and may later cause problems in conformance checking.

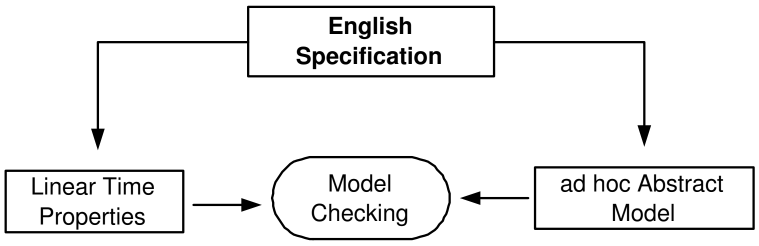


Figure 1. State of the art to capture a formal model

In [8] we developed Extreme Formal Modeling (XFM) to overcome the problems and restrictions of common methodologies to build formal models. XFM exploits the advantages of Extreme Programming (XP) in order to capture formal specification into abstract models. Figure 2 presents XFM's incremental approach to formal modeling. From the English specification, we first derive a simple formal property and then build an abstract model for this property. This is then model checked to verify whether the property holds for the model. Once the property is satisfied, we take a second property, extend the model according to this property, and model check for both properties. This procedure is repeated until the abstract model contains all behavior from the English spec. Simulating the model is a way to ensure that all properties defined in the specification have been accounted. The controlled and incremental model building results in a compact, structured abstract model. Whenever a property fails to validate, it usually is straightforward to find the bug as it must be related to the latest additions. The complete effort of modeling and bug fixing grows linearly along with the size of the model.

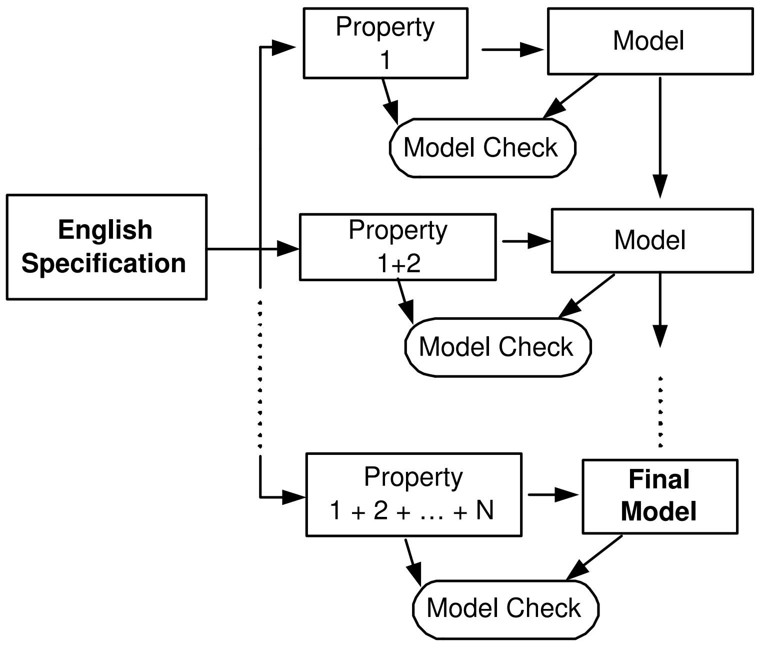


Figure 2. Capturing a formal model with XFM

To put this methodology to work we chose SPIN and PROMELA [15] as a vehicle because we want to be in the Linear Time properties framework rather than branching time. SPIN is probably the most popular model checker for linear time but we also used it because it can not only verify properties but also simulate the model, which significantly supports the debugging process. Other than that we use the tool LTL 2 BA [4] to visualize the LTL properties as finite state machine (FSM). This helps to interactively create LTL properties but also to detect errors. In order to show and explore the power of this new methodology, we develop the control of a Smart Home system which is an important example for ubiquitous computing.

1.1 Motivation for the Smart Home example

As technology advances, the cost and size of electronic components are becoming cheaper, smaller, and consume less power. As a result, these components are becoming an increasingly important part of our environment. Ubiquitous computing is no more science fiction, but instead an emerging technological area. There are electronic devices such as sensors, cameras, personal assistants, and microprocessors that provide us with information and transparently perform tasks without our knowledge of its working. This is what makes our environment "smart". A Smart Home uses these techniques to make living conditions more convenient and adapts to its residents' needs. A truly "smart" home does not directly burden its residents with technology, but reduces the presence of technology by automating tasks of everyday life. However, for "smart"-ness, the technology must be constructed such that the resident is unaware of the complex technology behind the automation. So, one important objective of a Smart Home is to take charge of obvious and repetitive tasks. However, complex tasks such as learning the resident's working schedule in order to automatically adjust the room temperature according to their arrival from work can also be performed. Although the need for explicit control is taken away, the resident has supervisory control over the technology to accommodate for changes that may occur in their schedules. For example, sudden changes in work schedules need immediate alterations to the technology's settings. Hence, it is important build a system where the user can change the setting/options according to his/her preferences at any time without much difficulty. Therefore, a good user interface is an important part of the Smart Home that allows the user to set the preferences accordingly.

It is desirable for a person that the environment is aware of his/her presence, acts according to his preferences, and is able to communicate with him easily in many ways. But while people are used to and accept that programs on their PC sometimes crash or behave inadequately, it is unacceptable that their home environment shows any unwanted behavior. It is unthinkable that a house refuses access to its resident or that vital functions such as heating, lighting, or the security system do not behave the way that they should. On account of this the development of the control for a Smart Home not only has to be done with diligence, but it has to include a methodology that completely rules out such failures. Extreme Modeling is a methodology that not only provides for a correct model of the system, but also makes the process of model-building and capturing of formal specifications faster and more intuitive.

1.2 Organization

This paper is organized as follows: In Section 2, we discuss related work, in Section 3 we present our modeling approach, the tools, and how they are used. Section 4 involves the description and implementation of the case study of the Smart Home. It includes all the important properties that are verified and how we used Extreme Modeling to construct the abstract model of the Smart Home. Finally, we conclude the paper with Section 5.

2 Related Work

Efforts are being made to use the methodology of extreme programming by many people to build stronger and powerful systems, but this approach has not been used much for formal modeling and verification. Some related work in the field of connecting formal methods and XP had been done by Herranz and Moreno-Nacarro in [14, 12, 13]. They describe the integration of some XP practices to formal methods using the SLAM software tool [5]. Their environment generates sequential programs from formal expressions using an assertion based JAVA development framework. While our work involves the use of XP to model complex concurrent hardware systems their approach is directed towards sequential software programs. General information about XP and agile techniques can be found in [18, 3, 7, 17].

In this paper, we formally model a Smart Home system and verify its properties. A lot of research [9, 16] as well as implementation models of Smart Homes exist. The OXIGEN project of MIT [6] focuses on pervasive human centered computing and an intelligent room. For our example model, we borrow parts from prototypes such as the Cisco Internet Home [2] or the Adaptive House of the University of Colorado [1]. The Internet Home built by Cisco involves security, remote monitoring, and control of appliances. The Adaptive House built by the University of Colorado also contains appliances, entertainment centers as well as temperature and lighting control units. However, it has the approach of a non-invasive technology, where usage patterns are learnt over time and then successively automated. This is in contrast to the Cisco Internet Home where the control is fixed and the focus is on the ability to remotely control the home environment. Our model of the Smart Home features some of the above mentioned properties and exemplifies the usage of formal and agile methods for such high reliability applications.

3 Modeling Approach

3.1 Linear-Time Properties

Linear Time Logic (LTL) is the leading specification language technique for temporal rules [10]. It extends propositional logic with the four operators “always” (condition holds always in the future), “eventually” (condition holds sometime in the future), “next” (condition holds in the next cycle), and “until” (condition A holds until condition B, afterwards do not care). All LTL operators are listed in Table 1. As to the expressiveness of LTL, it is complete with respect to first order logic [11]. Temporal expressions that cannot be expressed in LTL, can be provided to SPIN in the form of a never-claim automaton.

Table 1. LTL Operators

!A	negation
A→B	implication
A ↔ B	equivalence
A && B	and
A || B	or
[]A	Always
A U B	until
<> A	eventually
X A	next

Testing for the completeness of a LTL property can be one of the challenges of formal verification. Small changes in the LTL property, like misplacement of a parenthesis can change the complete meaning of the property. For example [](a→<> (b U c)) represents a simple 2-state automaton, saying that always if a occurs, eventually b will occur and stay true until c occurs. If we move the parenthesis before the <>, it is a 7 state automaton, and it is not easy to describe its behavior. Even if these kinds of mistakes are hard to detect, it is especially important that properties are correctly defined before running the model checker to verify it. If we check the model against a wrong property, it is highly probable to introduce more bugs into the model. A tool called LTL 2 BA [4] (LTL to Büchi automata) generates a Büchi automaton representing any LTL expression. This visualization is instrumental in verifying that the expression matches the specification. LTL 2 BA also generates PROMELA code from an LTL expression. Therefore it could - in theory - be used to obtain the abstract model directly and automatically from the LTL properties. However, in practice this does not work since first of all it is not possible to describe some implementation details such as initialization of variables and changing state variables and second, as it is possible to concatenate several properties with && this is not practical for a large number of properties since the resulting automaton may be very big. This is because LTL 2 BA is building a monolithic automaton, where the concurrency of independent parts is not captured. For these cases it is better to keep independent processes in the model as this relects better the structure of the system.

3.2 Extreme Formal Modeling

Rules of XFM are based on rules of XP. Many of the XP rules can be applied directly and successfully in XFM. Forinstance one of the main XP rules is to write tests before the actual code. In XFM this rule maps to specifying the LTL property before writing the abstract model. Another important XP technique is to add functionality as late as possible, keeping the model simple for as long as possible. Iterations are small steps in the development process. At the start of each iteration the goals are identified and written down in the form of "user stories" - individual cards that point out specific implementation details and requirements. These user stories act as a detailed guideline for the programmer. To refactor problems as much as possible, to update tests after a bug is found, and to work in pairs are also principles that are as beneficial to the capturing of formal methods as they are for common programming projects. The benefit of other XP techniques such as a stand up meeting in the mornings, collective code ownership and moving people around depends on the type of the project, the size of the team, and on personal and corporate preferences.

On the other hand, similar to XP, Extreme Formal Modeling also involves initially writing English specifications which can be thought of as user stories. These user stories are further broken down to Linear-Time properties which are checked for correctness by tools that convert these to automata. These properties are checked if they express what is intended, then a model is constructed based on these properties. Once the abstract model is constructed and the property is satisfied, we take the next property and follow the same procedure. If the property is found to be unsatisfied, we can locate the error with the help of the trace file generated by the model checker, fix the bug and rerun verification. Once verification is successful we pick the next basic user story, transform it into an LTL property and extend the model obtained from the previous property to satisfy this one as well. At each incremental step, all the properties previously verified are checked as well to ensure that the model maintains the behavior specified by all the properties. This procedure is repeated until the abstract model contains all the specified behavior from the English spec. We can also ensure that all functionalities of our English specification are incorporated in our final model by simulating the model. If a certain functionality is found missing, we identify the corresponding LTL property and extend the model accordingly. After correct simulation, the model and the list of LTL properties should be complete. This controlled and incremental model building procedure results in a compact, structured abstract model. This abstract model does not contain any unnecessary behavior.

The major benefits of our approach are the speedup of the model-building process and the high quality of the model compared with the traditional approach. Since we handle small steps, each step will add limited functionality to the model, so the debugging process is much more directed.

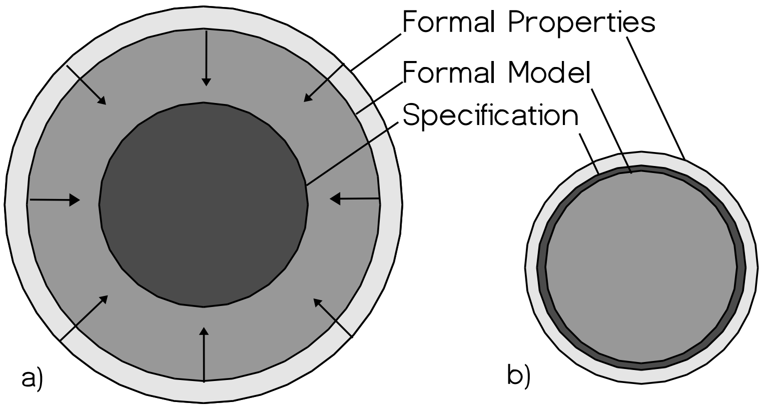


Figure 3. Modeling process (a) and modeling result (b)

Another benefit is that our model is built using the properties. Hence, it does not include much unintentional details. Figure 3 illustrates how the amount of behavior for the properties and the abstract model develop during the capturing process. At any point, the behavior of the formal properties is more general than that of the abstract model. At the beginning, however, they are both much more general than the behavior of the specification. In each iteration step their behavior is confined by adding additional properties and details to the model. Obviously the behavior of the specification does not vary during this process since the specification is not altered. Ideally, in the end all three behaviors are identical with the specified behavior, but in practice there will always be a small gap. However this gap will be much smaller for the XFM methodology than for the traditional approach. The fact that the behavior of the model is closely linked to the properties, entails a close to complete set of properties once the model is complete; simulation of the model will help reveal missing functionality. In the conventional approach, however, the model tends to contain much more functionality than specified, but less properties than needed as there is no mechanism that guarantees the exposure of all properties of the spec. The overall time to build and validate the model is substantially less, especially for large systems. This is mainly due to the iterative aspect. Since the model is checked for each property after each iteration, the time needed to debug is less. This is in contrast to debugging the entire model at once for satisfying all properties as per the traditional approach. For example if ad hoc models are built, they are usually monolithic, but with XFM, complex models get broken down to small problems and can be built as concurrent state machines more easily. The time required to build models using this methodology grows linearly in the size of the model, whereas the design effort in a conventional methodology grows exponentially with the size.

4 The Smart Home Case Study

4.1 General Properties

There are many approaches to what a Smart Home looks like and to what it should be able to do. The most important categories include the control of lighting, temperature, security, and safety. But some also include other features such as entertainment and specific smart appliances such as a smart fridge or a smart microwave. Systems also differ greatly in their level of awareness of the resident, where a higher level of awareness enables the system to adapt to personal preferences of different family members depending on their current location in the house and the time of day. Finally a big design decision is the level and way of interaction with the system. This may be as simple as a central control console or it can be done with multiple ways such as portable devices anywhere in the house, voice recognition, and remotely over the Internet and by phone.

4.2 Overall English Specification

The example of a Smart Home we are describing here includes all basic functionalities as well as extended features featuring advanced control. We describe the functionalities for the different categories although it is not possible to do a clear separation as some functions require interaction of several categories.

4.2.1 Lighting.

Lighting control will illuminate rooms depending on the the brightness outside. There are light sensors at every window. If during the day it is brighter than a pre-defined value, light in the room will switch off. Once it gets dark, the light will switch on. Obviously rooms without windows such as the bathrooms do not have light-sensors. Every person carries an RFID (Radio Frequency identification) in their jewelry or shoes depending on the type of RFID, and the lights in the house are controlled depending on the presence of persons in the rooms. If a room has not been used for more than 10 minutes, light will switch off. In addition for some rooms such as the living rooms and the bedrooms, the resident can choose an intensity from 0 to 100 percent. Whenever he comes back to the room, light should switch to the previous selected intensity. However, the next evening, as the light switches on when it gets dark outside, it should always be on 100 percent intensity.

4.2.2 Temperature Control.

Temperature control involves temperature sensors in every room, and the control of heating and air conditioning. For each room, three temperature levels can be defined: comfort level, power-save level and the vacation level. The comfort level is the temperature the resident likes to have when the room is being used. The second level is the power-save level that defines the temperature for a room that is not being used. It saves power by lowering AC and heating, but still keeping the temperature at a level so that the room can reach comfort level in a reasonable time. The vacation level defines the temperature for a room that is not used for a longer period of time. In the vacation level, only heating is operated to prevent the room from freezing. Whenever a person is detected in a room, the system goes to comfort level. If a room has not been used for a certain time for example 2 hours, temperature will be lowered to power save level. If the room has not been used for say more than two days, it will fall back to vacation mode.

4.2.3 Security.

Security uses motion sensors in every room to detect if someone is present in a room or not. If there is motion detected but no person of the household identified in the past 5 minutes, an alarm signal is issued. When the alarm signal is issued, the safety alarm starts off and a message is sent to the resident's cellphone along with an email. If the alarm is not turned off within next 5 mins, a signal is sent to the security for help.

4.2.4 Safety.

Safety is dealing with malfunctioning devices which may cause unforeseen hazards to the house such as fire, flooding etc. It may also involve mistakes made by the residents which may cause such an event. One of the main components of safety is the smoke detectors which can be installed in every room to monitor smoke levels. Also it checks if the temperature is in a safe range. Once a certain safety constraint is violated, a safety alert is raised and displayed on the screens available in every room. Also a message is sent to the residents phone and email box. If a hard safety constraint is violated, fire extinguishing sprinklers are activated and the fire department is notified.

4.3 Capturing of the Formal Model

In order to capture the formal model we identify a basic functionality of the system, specify a formal linear time property for this functionality and then build a model that satisfies this property. Our first property states that if it sufficiently bright outside, the light will never switch on in the room. Table 2 shows the corresponding LTL property. To make sure that this property expresses what we intend to express, we enter the LTL formula into the LTL 2 BA tool in order to get the corresponding automaton. Figure 4(a) shows the FSM corresponding to property 1 and Table 3 lists the definitions necessary to verify it with SPIN. Since this property is quite simple it is not difficult to see, that it expresses the correct behavior, but for more complex properties it is important to make sure they are correct before checking the model for it. Now we start writing a model in PROMELA that satisfies exactly this one property.

Table 2. LTL Properties for the Smart Home model

1	If light is over 500 lumen light will not switch on	[](klact → kbright)
2	If the room is not used for more than 10 minutes, the light is always off	[](kaway10m → !klit)
3	If there is light in the room, either it is sufficiently dark outside and someone is using the room or someone has been in the room in the last 10 minutes	[](klit → (khere10m && klact))
4	If the light is off, either it is sufficiently bright already, or it is just getting sufficiently bright or nobody has been in the room recently.	[](!klit → (!klact || X !klact || kaway10m))
5	If no identified person is in the room and there is motion the alarm is triggered	[](((kempty && kmotion) U alarm) || !kmotion || (kmotion && !kempty))
6	Temperature is at comfort level if the room has been used within the last 15 minutes	[](ktcomfort → khere15m)
7	Temperature is at power save level if the room has not been used for more than 2 hours.	[](ktpowersave → kaway2h)
8	Temperature is at vacation level if the room has not been used for more than 2 days.	[](ktvacation → kaway2d)
9	At dawn (when the light gets active) intensity is always at 100 percent.	[]((!llact U lintmax) || llact)
10	If there is some smoke or a high temperature in any room next a firehazard is issued	[]((lowsmoke || temphigh) → X safetyhazard)
11	If there is heavy smoke or a very high temperature in any room next the sprinkler system is started	[]((heavysmoke || tempveryhigh) → X spkl)

It is important to try not to introduce functionality early, however, the model always contains parts that have no correspondence in a formal property. This is because certain things such as initialization of variables and changes of state variables can not be expressed in linear time properties. Figure 5 shows the PROMELA model for the first property. It features the initialization process *init* and the process *KLuminosity*, where the variable *kactive* is set according to the current luminosity. In order to keep the state space of the model small, we only consider three values of brightness. All other values will not result in any change of *kactive*.

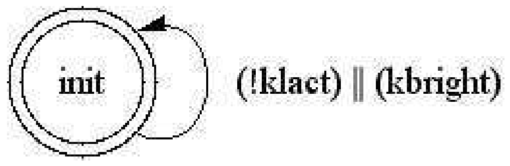
Only after the first property is verified, we come up with a second property. This property states that if the room has not been used for more than 10 minutes, the light is always off. The LTL property in Table 2 is similar to property 1. A process *KPsense* has to be added to to model that detects if a person

Table 3. Definitions for the LTL Properties

1	#define kbright klum $<$ 500	Brightness level in the kitchen is under 500 lumen
2	#define klact klactive	True if kitchen light can be switched on, false if bright enough
3	#define klit klight	Output for the kitchen light.
4	#define ktcomfort ktl==comfort	Temperature in the kitchen set to comfort
5	#define ktpowersave ktl==powersave	Temperature in the kitchen set to power save
6	#define ktvacation ktl==vacation	Temperature in the kitchen set to vacation
7	#define kempty kpsens==Empty	No person is in the kitchen
8	#define alarm alert	
9	#define khere15m kcount$<$16	
10	#define khere10m kcount$<$11	Kitchen has been used within the last 10 minutes
11	#define kaway10 kcount $>$ 10	Kitchen is unused for more than 10 minutes
12	#define kaway2h kcount $>$ 15	Kitchen is unused for more than 2 hours
13	#define kaway2d kcount $>$ 20	Kitchen is unused for more than 2 days
14	#define kpsense !(kpsens == Empty)	No person detected via RFID
15	#define kmotion kmot	The motion detector
16	#define lowsmoke ((ksmoke==LOW)||(lsmoke==LOW))	There is some smoke in a room
17	#define heavysmoke ((ksmoke==HIGH)||(lsmoke==HIGH))	There is heavy smoke in a room
18	#define temphigh ((ktemp $>$ 80)||(ltemp $>$ 80))	The temperature in a room is unusually high
19	#define tempveryhigh ((ktemp $>$ 100)||(ltemp $>$ 100))	The temperature in a room is very high
20	#define spkl sprinkler	The sprinkler system
21	#define lintmax lint == 100	Full light intensity in the living room

is in the kitchen and it counts the minutes since the kitchen has been used last. If nobody has been there for more than 9 minutes, the light is switched off.

Property 3 describes that if there is light in the kitchen, it must be sufficiently dark outside and someone used the kitchen within the last 10 minutes. The LTL property is quite easy to understand. For the formal model, we only have to add small changes to reflect this property. Basically we have to make sure that the light is dependent on *kactive*. Once these changes are performed, the property verifies and we determine the next property. Number 4 defines when the light is off in the kitchen. If the light is off, that means, that either nobody has been using the room recently or it is sufficiently bright without light. This property is a good example for the interactive model building process. While implementing the changes in the model we realize that when it is just getting bright in the morning, the light is switched on just



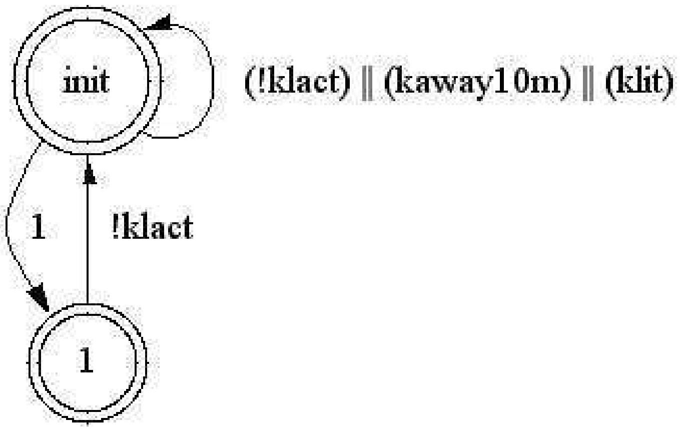


Figure 4. FSMs of properties 1 (a) and 4 (b)

before *klactive* is switched off. This will cause the property to fail. Now if we inverse the assignments, property 3 will fail. This is why we extend the property including instants where in the next state *klactive* is true. To make sure, that this actually happens in the next state we have to enclose the two assignments in an *atomic* statement. Figure 4(b) shows the corresponding FSM generated by LTL 2 BA. It has very few transitions but close examination confirms that the automaton follows our intention. When building the properties independent from the model, such details get omitted. This will cause, the property to fail and it may be time consuming to locate the error.

As the model now contains most of the lighting functionality, we add a security property next. Property 5 exploits the motion sensors. When no person is identified in the room with the a valid RFID and there is motion detected an alarm is activated. The alarm signal might ring a bell and send a text message to the residents cell phone and email. As the alarm signal can only occur after the condition is detected, it is most appropriate to use the until operator U for this property. Table 2 shows all the LTL properties. The implementation in the formal model does add some lines to the *KPsense* process, but it does not incur any fundamental changes.

Next, we consider the temperature control mechanism. As there are three basic properties that are similar and closely related, we decide to add all three properties in one iteration step. The properties say that the AC/Heating system has three modes of operation, each having a temperature level is assigned. The first level is the comfort level, it defines the temperature when the room is used. Then there is a power save temperature that is active whenever nobody has been using the room for more than two

```
bool kactive;
int klum;
proctype KLuminosity()
{
  do
    :: klum < 400 -> klum = 200; kactive =1
    :: (100 < klum) && (klum < 700) -> klum = 400; kactive =0
    :: klum > 700) -> klum = 600; kactive =0
  od;
}
init
{
  klum = 200;
  kactive =1;
  run KMotion();
}
```

Figure 5. Promela code for first property

hours. And finally there is a vacation mode that is active when a room has not been used for more than two consecutive days. The resulting LTL properties are simple (Property 6, 7, 8). In the PROMELA model there is already a counter that counts the time since the room has been used last so we have again few changes in the model. After modifying the model, small corrections have to be performed when checking all previous properties.

Until here, our model only comprises one room, the kitchen. As we have sensors for every room, most of the control is also based on the room-level. All properties so far can be replicated for any other room. The same is valid for the model. This makes our model modular in approach. However, in some rooms there are additional functions that should be added. For example we want to control the intensity of the light in the living room and in the bedrooms. Intensity in the bedroom will be set to zero, when going to bed and in the living room it may be set to a desirable for example for watching movies. The fact alone to have a switch that controls the intensity will result only in trivial LTL properties, and therefore is not worth checking for. But in the specification, it says, that if it gets dark the next day, intensity should always be at 100. This requirement is expressed in LTL property 9 (see Table 2). To reflect this in the model, we first of all have to create the living room. This is done by copying all kitchen processes and rename them and the variables. In addition to *KPsense* we get *LPsense* and so on. Then we replicate all LTL properties and the definitions and check all these properties as well as the ones for the kitchen. Once this is done a new process *LIntensity* is added, that switches between different levels of intensity if it is sufficiently dark in the living room. In order that property 9 verifies, we change the intensity back to 100 as soon it gets bright enough outside, as soon as lactive is turned to zero.

As we have several rooms now, it makes sense to introduce functions that spawn several rooms. Safety properties such as the detection of smoke or hazardous temperatures are properties that concern the whole house. In order to ensure the safety in the house, we add two properties, that depend on the smoke detectors and on the temperature sensors. Property 10 issues a safety hazard when a certain amount of smoke is detected or if the temperature in any room is high or low. A safety hazard will notify the resident about the incident but not take any immediate action yet. Once the intensity of the smoke reaches another critical level, the sprinkler system is launched and the fire fighters are notified. In the model, these changes induce the creation of two small processes that monitor the temperature and smoke sensors in all rooms.

5 Conclusion

In this paper we demonstrate the usage of the XFM methodology developed by us in [8] for the formal modeling of a large control application for a smart building. It focuses on the concept of incremental formal modeling based on properties from the English specification. Each property represents a specific behavior based on which the abstract model is constructed. The incremental approach used in the paper guarantees that the constructed abstract model not only satisfies all properties, but also ensures that it contains only little unwanted behavior that might cause the implementation to fail at a specific step. Since XFM involves an iterative technique, the evolving abstract model facilitates debugging whenever a property is found unsatisfied. In each iteration step the behavior is confined by adding additional properties and details to the abstract model. The fact that the behavior of the abstract model is closely linked to the properties entails a close to complete set of properties once the abstract model is complete. In the conventional approach, however, the abstract model tends to contain much more functionality than specified, but less properties than needed as there is no mechanism that guarantees the exposure of all properties of the spec. The overall time to build and validate the model is substantially less, especially for large systems due to the iterative modeling aspect. We chose the SPIN model checker as modeling environment not only because it is one of the most popular model checkers in todays modeling of concurrent systems, but also due to the fact that it comes with a convenient simulator to debug the model when properties fail.

The modeling example of the Smart Home demonstrates the power of the approach, and even if there exist more sophisticated smart spaces, we succeeded in building a reasonably complex smart environment with little effort. Yet most important is not the large amount of functionality and the different levels of interaction, but the confidence that this model fully complies with the specification without containing much superfluous behavior.

References

[1] Adaptive home- colorado, boulder. http://cs.colorado.edu/˜mozer/nnh/index.hmtl.

[2] Cisco internet home. http://www.cisco.com/warp/public/3/uk/ihome.

[3] Extreme programming: A gentle introduction. http://www.extremeprogramming.org/.

[4] LTL 2 BA : fast algorithm from LTL to buchi automata. http://www.liafa.jussieu.fr/˜oddoux/ltl2ba/.

[5] The slam website. http://lml.ls.fi.upm.es/slam.

[6] Mit project oxygen - pervasive human centered computing. http://oxygen.lcs.mit.edu, 2003.

[7] Kent Beck. *Extreme Programming explained: Embrace change*. Addison Wesley, 2000.

[8] David Berner, Syed Suhaib, Sandeep Shukla, and Harry Foster. XFM: Extreme Formal Method for Capturing Formal Specification into Abstract Models. Technical Report no. 2003-08, Virginia Tech, FERMAT Lab, Blacksburg, VA, 2003.

[9] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In Paddy Nixon, Gerard Lacey, and Simon Dobson, editors, *1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, pages 201–212, Dublin, Ireland, December 1999. Springer-Verlag.

[10] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier Science Publishers, Amsterdam, 1990.

[11] Jesper G. Henriksen. *Logics and Automata for Verification - Expressiveness and Decidability Issues*. PhD thesis, Basic Research in Computer Science (BRICS), University of Aarhus, Denmark, June 2000.

[12] A. Herranz and J.J. Moreno-Navarro. Formal agility. how much of each? In *Taller de Metodologias Agiles en el Desarrollo del Software. JISBD 2003*, pages 47–51, Alicante, Espana, 11 2003. Grupo ISSI.

[13] A. Herranz and J.J. Moreno-Navarro. Formal extreme (and extremely formal) programming. In Michele Marchesi and Giancarlo Succi, editors, *4th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2003*, number 2675 in LNCS, pages 88–96, Genova, Italy, May 2003.

[14] A. Herranz and J.J. Moreno-Navarro. Rapid prototyping and incremental evolution using slam. In *14th IEEE International Workshop on Rapid System Prototyping, RSP 2003)*, San Diego, California, USA, June 2003.

[15] Gerhard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison Wesley, Boston, MA, September 2003.

[16] Stephen Peters and Howie Shrobe. Using semantic networks for knowledge representation in an intelligent environment. In *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*, Ft. Worth, TX, USA, March 2003. IEEE.

[17] Laurie Williams. The XP programmer - the few minutes programmer. *IEEE Software*, 20(3):16–20, May/June 2003.

[18] William A. Wood and William L. Kleb. Exploring XP for scientific research. *IEEE Software*, 20(3):30–36, May/June 2003.