

Automatic Acquisition of Task Models From Object Oriented Design Specifications: A Case Study

Shijian Lu & Cécile Paris

*CSIRO Mathematical and Information Sciences, Locked Bag 17, North Ryde, NSW 1670,
Australia,*

{shijian.lu, cecile.paris}@cmis.csiro.au

Abstract

Task analysis and modelling have been successfully employed in training, design evaluation, and user documentation productopn. However, building task models from scratch can be a time consuming process. In principle, it is possible to automatically acquire draft task models from system behaviour models defined in object oriented design specifications based on the common semantic ground between the two models. The work presented in this paper is a continuation along this line in order to investigate the feasibility and practicality of this approach. To this aim, a case study is conducted on the BMS system which was designed and developed by a commercial company. Apart from describing problems encountered and lessons learned, we attempt to draw some guidelines to encourage a more disciplined approach to system behaviour modelling.

1. Introduction

As software systems become increasingly sophisticated and complex, their usability becomes evermore important. While object oriented (OO) approaches to software development are good at delivering quality architecture, they are much less effective at ensuring usability. Despite of the consensus that task analysis and modelling should improve the resulting products' usability if properly integrated into the OO design process, the uptake of task analysis and modelling in OO development process is still rare. Intuitively, there are a lot of similarities between task models and system behaviour models, use cases, in particular [2]. In order to effectively integrate task analysis and modelling into OO design process, we need to know how to feed the result of task modelling into subsequent system design [8]. That, in turn, demands a good understanding of the relationship between task models and system behaviour models. Studies have shown that system behaviour models are semantically equivalent to task models [5]. However, in many aspects, task modelling appears to be better positioned to address usability issues than use cases [1], as they are currently defined in UML [7].

The common semantic ground between system behaviour models and task models has been exploited to provide a mechanism to integrate task modelling into OO design process [4], to enhance current OO CASE tools [3], and to construct draft task models automatically from OO design diagrams such as use cases, use case diagrams, and sequence diagrams [5] in order to generate user documentation [6]. Given that building task and domain models from scratch can be a time consuming process, it becomes important to know how automatic task model acquisition works in practice. To this aim, a case study is conducted on the BMS (Building Management System) which was designed and developed by a commercial software company. Apart from describing problems encountered and lessons learned, we attempt to draw some guidelines to encourage a disciplined approach to system behaviour modelling.

2. The BMS case study

2.1 Background

The BMS system is a building management system which is used by four user groups: the building owner, the business tenant, individual employees, and the building supervisor. The building owner lets office space to business tenants in the form of zone or section. A zone comprises a group of rooms, while a section consists of a number of zones. Once a zone or section is rented, the business tenant may establish climate conditions (profiles), which will be in effect within the rental period.

A business tenant controls the climate of the office space through the use of set points which define the desired lighting and temperature of the rented zone or section. The climate of rented rooms is normally collectively controlled at the zone level. However, there are two external systems, the Heating, Ventilating, And Cooling (HVAC) system and the Lighting System, which can be used for climate control at the room level.

An employee working in a room that is uncomfortable may request the building supervisor to release his/her room from zone control. In that case, the employee is able to set the climate conditions desired for his/her own room.

The BMS system was designed with the support of Rational Rose and implemented in Visual C++. Figure 1. shows the BMS system's user interface. In this study, first, a task model was generated from the BMS design model by using our Task Model Acquisition Module[5], which is written using the script language provided by Rational Rose. Second, the task model for using BMS was manually created by experimenting with the use of the actual BMS system. Finally, the result of step 1 and step 2 are compared and analyzed.

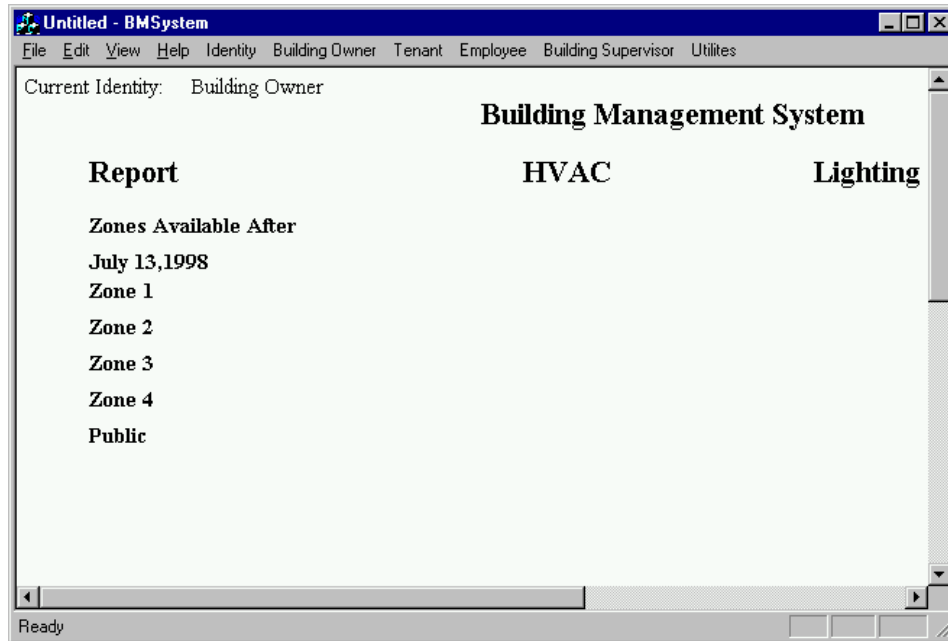


Figure 1. BMS system user interface

2.2 The original design model and corresponding task model

Let us first have a close look of the BMS design model. Apart from a class diagram, it contains 1 use case diagram (Figure 2.), 16 use cases, 15 sequence diagrams, and 6 collaboration diagrams. Examining the model in detail, the following shortcomings were found:

- Unclear use case naming: use cases were not clearly named. In principle, all use cases should start with a verb followed by a succinct phrase which will state the goal of the use case. For example, instead of “Room Occupant Uncomfortable”, “Change room setting” should be used (ref. Section 2.4).
- Incomplete design model: Use cases are high level tasks which are important to the system. Therefore, all use cases should be elaborated by sequence diagrams. However, this was not the case with the BMS model. For instance, there is no corresponding sequence diagram for the use case “Business Rents Additional Zone”.
- Missing actors: All actors are missing in the use case diagram. It is important to associate actors with their use cases especially when there were multiple user roles as is the case in the BMS system. The reason is obvious: different roles have different responsibilities and often entail different accessing privileges.
- Flat-structured use case organisation: use cases are equivalent to composite tasks. They naturally form a hierarchy according to different levels of abstraction. Therefore, they should be organised hierarchically (ref. Section 2.4) rather than as a flat structure as it was the case in BMS.
- Incomplete (or missing) user interface objects: user interface objects were not modelled in BMS. As a result, users were depicted as directly interacting with in-perceptible objects, e.g., in Figure 3, the “Tenant” was interacting directly with “Climate Set Point”, which is an internal system object.

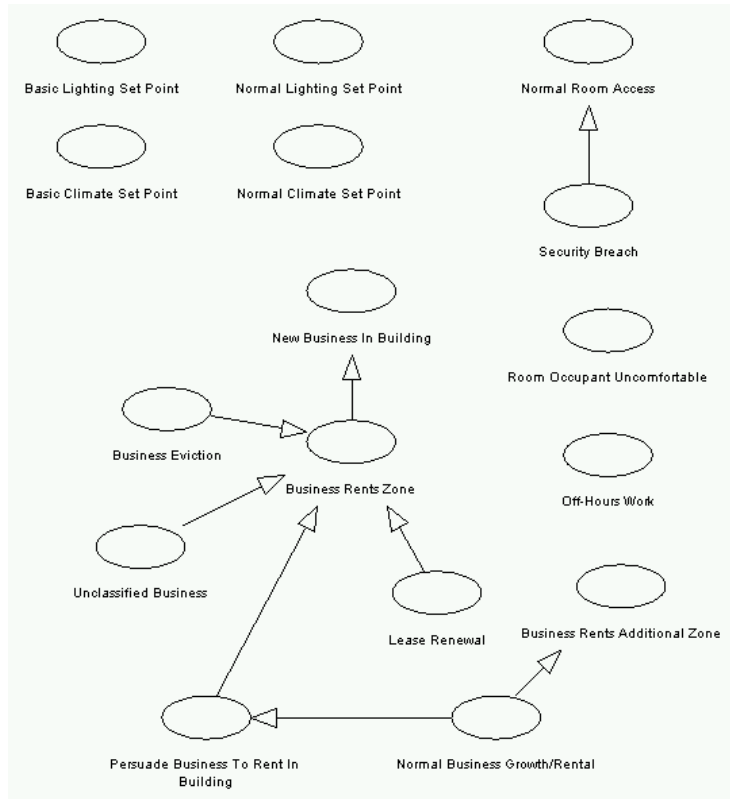


Figure 2. BMS use case diagram

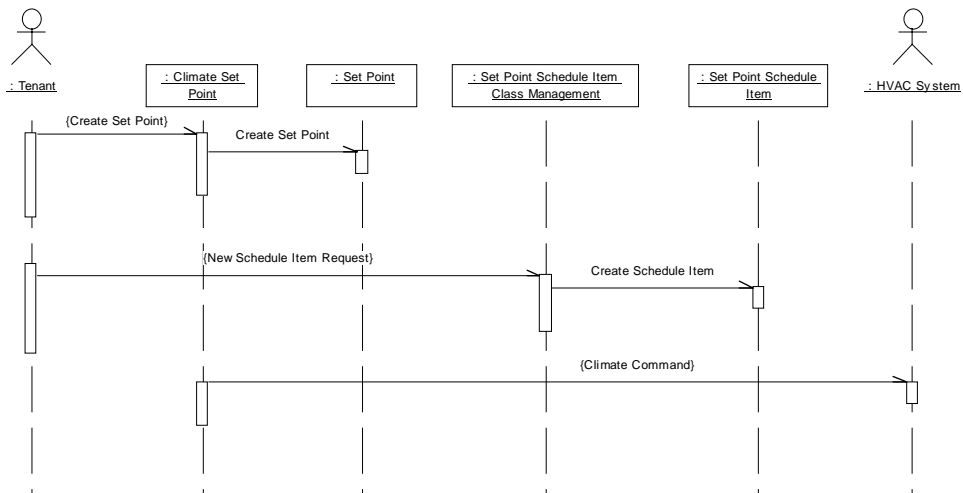


Figure 3. The interaction diagram for “normal Climate set point”

In the BMS design model, the use case “Normal lighting Set Point” was elaborated by the interaction diagram shown in Figure 3. A draft task model was automatically generated from the BMS design model. The task model shown in Figure 4 corresponds to the use case diagram shown in Figure 2, while Figure 5 corresponds to the interaction diagram in Figure 3. It is worth noting that, “Extend” relationships between use cases in a use case diagram are equivalent to task

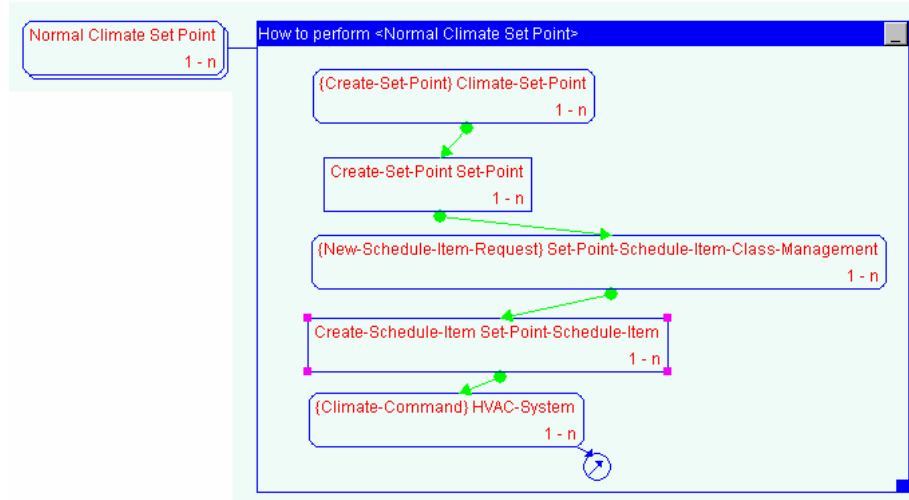


Figure 5. The task model corresponding to the interaction diagram in Figure 3.

2.3 Comparison analysis

Another task model for using BMS was manually constructed by experimenting with the use of the actual BMS system, as well as by referencing the actual implemented codes. For the sake of easy referencing, we call this task model *B*, while the one automatically generated from BMS OO design model, is referred as task model *A*. Briefly, task model *A* contains 16 basic composite tasks and all tasks belong to one of 3 levels whereas in task model *B*, there are 25 basic composite tasks and 4 levels. The extra level is introduced to differentiate tasks to be performed by different user groups. By comparing the two task models, we have the following observations.

- There are fewer elementary tasks in task model *A* than task model *B* due to the fact that user interface objects were only modelled in *B*.
- Apart from missing user interface object in interaction diagrams, inconsistency were found between what was modelled in the interaction diagrams and what was actually implemented. For example, “Add Set Point” in task model *B* shown in Figure 6 corresponds to task model *A* shown in Figure 5.
- Generally there is a big discrepancy between task model *A* and *B*. However, the task acquisition module works well in capturing the system behaviour model’s structure and contents. Nevertheless, a reasonable task model can only be generated from a reasonable design model.
- One problem discovered about the current form of the task acquisition module is that it does not address the issue of multiple user types.
- There seems to be a need for a disciplined approach to system behaviour modelling in object oriented design.

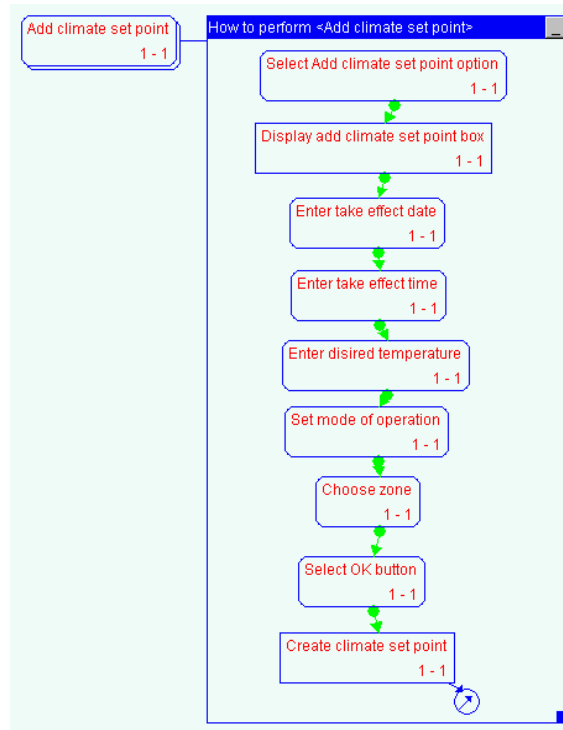


Figure 6. The final task model for “Add climate set point”

2.4 Recommendations for system behaviour modelling

Modelling task hierarchy

As we mentioned earlier, use cases come naturally in different levels of abstraction. Unfortunately, there is no explicit support for modelling this hierarchical structure in Rational Rose. This flat structure makes use case management difficult, especially when dealing with large projects. It also renders use case modelling harder to understand, and prevent easy identification of relationships between use cases. Drawing from our experiences, we have devised an implicit way to model hierarchical relationships. That is done by presenting the topmost level use cases in the use case diagram under the “use case view” package. The relationship between these use cases are presented in the diagram. If some or all use cases in the diagram are too complex to be handled as a single use case, they can be further decomposed into subordinate use cases. All subordinate use cases are represented in a use case diagram which should have the same name as its parent use case and be attached to that use case. The process will continue until a level at which the use cases are considered simple enough. At that point, the use case is elaborated by a sequence diagram which should share the same name as its parent use case.

Naming convention

To facilitate reuse, use cases, use case diagrams, and interaction diagrams should be composed of at most of 3 word units:

- Verb, describing the action
- Actee, describing the entity to be acted on
- Proposition, modifier of the action

Each unit, if consisting of more than one word, should be dash jointed together, e.g., Reassign room to-different-zoon. In the above example, “Reassign” is the Verb, “room” is the Actee, and “to-different-zoon” is the Proposition.

3. Discussion and conclusion

In this paper, we briefly presented a case study of automatically acquiring task models from object oriented design specification for a commercial software product. In particular, problems encountered and lessons learned. The study demonstrates that the quality of the resulting task model is determined by the quality of the object oriented design model. It has been shown that there were number of problems in the design model used in this study. Attempts are given to draw some guidelines or recommendations to encourage a disciplined approach to system behaviour modelling. While this study helps to shed some light on issues related to automatic task model acquisition, it does not suffice to draw a conclusion on how realistic the approach is. More case studies are needed to gain a further understanding of the current practice in industry as well as the practicality of a disciplined approach to system behaviour modelling.

Acknowledgments

This work is partially supported by the Office of Naval Research (ONR) – Grant N00014096-1-0465 – in the program for User Centred Direct Interaction Systems. We gratefully acknowledge the participation of all the members of our project team: Keith Vander Linden, Sandrine Balbo and Nadine Ozkan, and are thankful to Valery Anciaux and Christophe Plier for their contribution to the implementation of the task model editor.

References

1. Artim, J. M.; van Harmelen, M.; Butler, K.; Guliksen, J.; Henderson, A.; Kovacevic, S.; Lu, S.; Overmeyer, S.; Reaux, R.; Roberts, D.; Tarby, J.-C.; and Vander Linden, K.: "Incorporating Work, Process and Task Analysis Into Commercial And Industrial Object-Oriented Systems Development", ACM SIGCHI Bulletin, Vol. 30, NO. 4, Oct. 1998.
2. Artim, John M.: Integrating user interface design and object-oriented development through task analysis and use cases. CHI'97 workshop on Object Oriented User Interfaces. 1997, <http://www.cutsys.com/ooui/>
3. Lu, Shijian; Paris, Cécile and Vander Linde, Keith: "Enhancing CASE tools to support usability", in Proceedings of AWCSET'98, Australian Workshop on Constructing Software Engineering Tools, Adalaide, November 1998.
4. Lu, Shijian; Paris, Cécile and Vander Linde, Keith: Integrating task modelling into the object oriented design process: a pragmatic approach, position paper at the CHI'98 workshop on Incorporating Work, Processes and Task Analysis into Industrial Object-Oriented Systems Design, Los Angeles, May 1998, <http://gate.cruzio.com/~artim/CHI98/PositionPapers/LuParisLinden.htm>
5. Lu, Shijian; Paris, Cécile and Vander Linde, Keith: Towards the automatic generation of task models from object oriented diagrams, in the Proceedings of the IFIP Working conference on Engineering for Human-Computer Interaction, Crete, Greece, September 1998.
6. Paris, Cécile; Vander Linde, Keith and Lu, Shijian: "Automatic Documentation Creation from Software Specifications", in Proceedings of the Third Australian Document Computing Symposium, Sydney, August 1998.
7. UML (1997) Notation Guide: unified modelling language version 1.0, Rational Software corporation.
8. Van Harmelen, M. (1997) Idiom: an object-oriented user interface design methodology. CHI'97 workshop on Object Oriented User Interfaces. <http://www.cutsys.com/ooui/>