

Towards Software Product Lines Application in the Context of a Smart Building Project

Thibaut Possompès^{1,2}, Christophe Dony², Marianne Huchard², Hervé Rey¹,
Chouki Tibermacine², and Xavier Vasques¹

¹ IBM France – PSSC Montpellier

thibaut.possompes, reyherve, xavier.vasques@fr.ibm.com

² LIRMM, CNRS and Université de Montpellier 2

possompes, dony, huchard, chouki.tibermacine@lirimm.fr

Abstract. This paper presents proposals on how to apply software product lines in the context of smart buildings control systems. Our study is held in the context of the RIDER (Research for IT as a Driver of Energy efficiency) project, which is led by a consortium of several companies and research laboratories.

The paper highlights various issues, including issues related to traceability, variability and automatic transformations in our specific application context. It then proposes solutions to apply existing concepts like feature diagrams, model-driven transformations and component-based architectures, or appropriate extensions of these concepts, in this context.

Furthermore, we globally propose an approach to federate the various components involved in smart buildings, and to capitalize on the developed IT components by reusing them at the lower cost in the construction of future buildings.

Key words: Smart buildings, feature diagrams, traceability, model transformations

1 Introduction

Nowadays, buildings are responsible for huge energy consumption sometimes because of wrong, or not optimized machinery control. For instance, cooling an empty room during the summer is useless and represents an energy waste. Installing a presence detector would be an efficient way to detect when to engage the cooling system. Instrumenting buildings to track and solve wrong usage of energy management systems, will result in using energy more efficiently and therefore stop wasting it. Such instrumented buildings are called *smart buildings*. Their particularity is that they allow gathering information from different systems (*e.g.* Heating Ventilation and Air-Conditioning (HVAC) or room occupation agenda) to optimize their settings in order to enhance building energy efficiency.

In this paper we propose an approach to federate the various components involved in smart buildings, and to capitalize on the developed IT components

by reusing them at a lower cost in the construction of future buildings. We achieve this goal by using software product lines methodology for:

- gathering domain knowledge from the different stakeholders, and managing the whole software life-cycle of smart buildings related projects.
- modelling IT infrastructure necessary for instrumenting a building and analysing related data.

This leads us to describe how we could generate some parts of the IT infrastructure thanks to feature diagrams. We illustrate our approach in a model-driven perspective and use the principles of the model transformation ([1,2]). Another key aspect of our project is to enable traceability through developed IT artefacts, domain models and feature diagrams in order to be able to manage domain specific and IT concepts as a whole. This research is done in the context of the RIDER project³.

Smart buildings, or houses, are buildings instrumented with specialized equipment, *e.g.* sensors and controllers (as presented in the home automation example in [3]), in order to enable their optimization according to specified criteria. Some challenges associated with this domain consist in optimizing building management equipment operations to enhance the global energy efficiency, and reusing IT components from one building to another to reduce costs. This leads us to think of a specific building instrumentation as a product derived from a building product line.

Software Product Lines are defined by Pohl et al. in [3] as “a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customisation”. Using *platforms* requires implementing and reusing reusable software artefacts in order to fit to specific client needs at a lower cost. This helps building customisable applications and implies using a strict approach for identifying variations. This last concept requires being able to specify commonalities and variation points associated to each modelling artefact produced during software development life cycle, including requirements, architecture, components and tests artefacts.

Section 2 describes the project’s context. Section 3 presents how domain issues will be addressed thanks to software product lines concepts. We also describe which methodology will inspire us to solve challenges identified in the project and how we can gather requirements, specific domain vocabulary, domain models and software architecture. Section 4 discusses how will be concretely implemented software product lines in the project. This will be described in depth by specifying the software levels impacted by the product lines approach, the link between product line artefacts and the project IT architecture, and finally

³ The RIDER project (“Research for IT as a Driver of EnERgy efficiency”) is led by a consortium of several companies and research laboratories, including IBM and the LIRMM laboratory, interested in improving building energy efficiency by instrumenting it.

the link between the product line and the domain models. The conclusion and perspectives section sums up how model driven engineering approaches could be applied to software product lines in the smart buildings context.

2 Specification of the Smart Buildings Context

This section describes the specific constraints related to smart buildings and, more specifically, the RIDER project. We will present details of modelling an instrumented building.

2.1 Global Issues

Smart building energy optimization addresses various building management related activities such as energetics experts, Building Management System (BMS) manufacturers, electronics engineers, *etc.* Instrumenting such a heterogeneous domain to satisfy every stakeholder and to be able to furnish pertinent optimisation scenarios is a complex task. It requires a very close collaboration between partners and a very modular and reusable IT architecture.

This context raises several questions. How to unite partners around a common domain vocabulary described in IT models ? How to adapt IT infrastructure to interface with such large numbers of BMS, sensor types, building architectures ? Clients can have very different expectations from an instrumented smart building in function of their practical and financial requirements. Furthermore, the environment can affect the way a building can be optimized, *i.e.* buildings in Delhi will take advantage of different climate specificities than ones in Stockholm. How to represent and deal with such variations ?

Hence, software product lines looks to be a very appropriate approach in order to ease reuse of the IT optimization infrastructure deployed on a pilot building and to unite the heterogeneous points of view around a generic and extensible IT architecture.

The software product lines feature diagrams are trees representing the main functions of the system. They are understandable by all stakeholders thanks to their simplicity. This is a good medium to communicate ideas between them.

Another argument in favour of software product lines, as outlined in [4], is that it has a strong impact on quality of resulting software. Indeed, it is built upon mature and proved components which implies lower number of defects, hence the resulting systems are more reliable and more secure.

2.2 Excerpt of the RIDER Project Context

This project's purpose is to achieve, thanks to IT, the interconnection of several fields linked to smart buildings. It encompasses specific domain fields, such as energetics, building automation, electronics, *etc.* First of all this "intelligent" coordination of systems must be brought by IT through which we want to implement functionalities such as:

Possompès et al.

- Analysis of monitored data
- Visualization of monitored data
- Energy efficiency scenarios application

These features require having a model of the physical components (sensors and actuators) placed in the building and retrieving monitored data as shown in Figure 1. The monitored information stream is generally reported by a *BMS*

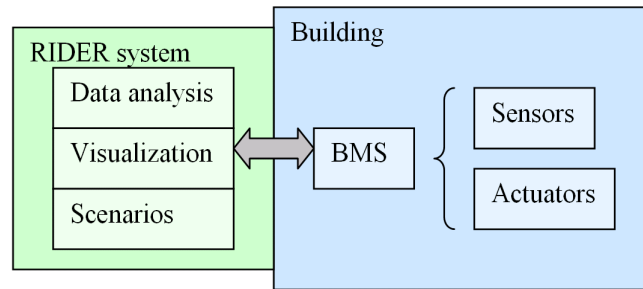


Fig. 1. RIDER data acquisition from a building

whose role is to automate basic building management functions based on the data acquired by the sensors.

To sum up, we want to assemble concepts coming from different domains related to buildings that were not communicating to each other, and to develop software components adaptable to the existing instrumentation facilities and communication interfaces.

3 Domain Analysis with the Help of Software Product Lines Engineering Concepts

As said before, many players are involved in smart building projects. Hence, it is necessary to have a good methodology to acquire information about what can change from one building to another.

Existing Product Line Approaches for Analysing a System As presented in FODA [5] all stakeholders must be involved in the three phases composing the domain analysis process:

1. *Context analysis* to define the domain context.
2. *Domain modelling* to describe the problems addressed by software in this domain. It also provides the features of the software, standard vocabulary of the domain experts, documentation and generic software requirements.

3. *Architecture modelling* to establish the structure of software implementation in the domain to help developers. This architecture can guide to build libraries of reusable components.

Features can be linked to artefacts created during each one of these phases. As described in FORM [6] features can be classified according to the type of information they represent. We classify features into the following categories, or layers:

- *Application capabilities features* characterize distinct services, operations, functions, and the performance that an application may possess (*e.g.* optimizing heaters accordingly to users preferences).
- *Operating environment features* represent the attributes of the environment in which an application is used and operated (*e.g.* operating system, software platform such as IBM Tivoli[®] used for managing and monitoring systems).
- *Domain technology features* represent implementation details, at a low and technical level, specific to a given domain (*e.g.* HVAC valve control).
- *Implementation technique features* describe technical details that could be used in other computer science areas (*e.g.* wireless sensor communication protocol).

Each of these layers will require the participation of the project stakeholders.

By using this classification, we can establish feature diagrams that will target specific descriptive needs. Furthermore, we will use the software product line framework cited in several books ([4,3]) and defined in [7].

We can observe that each layer can be associated to one or several framework steps. *Application capabilities* are close to *domain requirements* and *product management* steps. Figure 2 gives a small example of the kind of feature we can expect in the context of smart buildings. The feature called *RIDER* is the root feature (concept explained by Czarnecki et al. in [8]). We can derive from it two mandatory features named *Intelligence* and *Building management system interface*. *Intelligence* feature contains *Visualization*, *Data analysis* and *Building management scenarios* sub features representing three possible different ways to enhance the management of instrumented buildings. The *Building management system interface* feature encompass the basic functionalities necessary to interface the RIDER system to a *BMS*. Its two sub features shows that it is possible to acquire data from a building and to send orders to it. They are not mandatory but not choosing them would drastically restrict *Intelligence* sub features choice. Indeed, constraints relationships are not shown on this diagram but we can assume that *Data analysis* and *Building management scenarios* require the *Data acquisition* feature.

Operating environment and *Implementation technique* are related to *domain design* and *realisation*. Figure 3 shows a very global view of the RIDER architecture. Software products will be chosen to carry out the functionalities expected from each box. These products would be described in the *operating environment* layer ; the way information is transmitted from the BMS to the RIDER system would be associated to the *Implementation technique* layer.

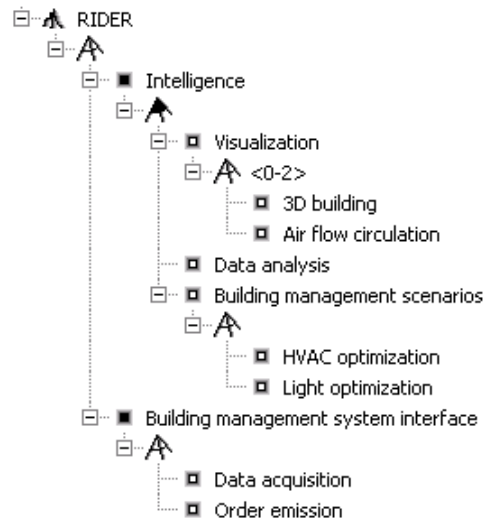


Fig. 2. Excerpt of a feature diagram related to buildings domain (all feature diagrams of this paper are created with Waterloo University feature modelling eclipse plug-in)

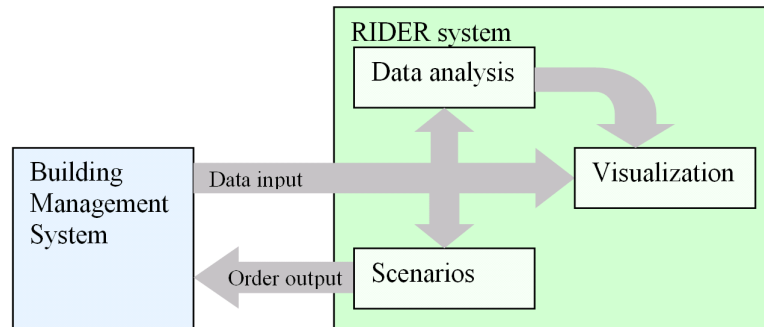


Fig. 3. RIDER architecture example

Feature diagrams would be a great help for allowing all different domain experts to express the differences between the requirements concerning the IT infrastructure thanks to a non IT language. Indeed, we want to keep a traceability through all steps by linking feature diagrams together in order to be able to check, *e.g.* whether client requirements are fulfilled by architectural choices.

Existing Approaches and New Propositions to Capitalize upon Domain Knowledge A further work in our approach would be to base domain models upon feature diagrams describing their semantics with domain vocabulary. This would help all stakeholders to understand and communicate with each other with a common language and make UML models semantics accessible to non specialist users. Hence we could link functionalities throughout several domain models and requirements specifications (*e.g.* building components like HVAC and thermal regulation concept).

Many enhancements have been made to the initial feature diagrams presented in FODA [5]. Fey et al. [9] introduced *feature sets* in order to group features from an arbitrary point of view. This can help domain experts to categorize the functionalities they expect from their domain model. Zhang et al [10] added features *binding time* property to specify when the feature must be either implemented or removed. We plan to extend this functionality to be able to check partially customized feature models to allow the different domain experts to choose the features that fit to their needs or practical possibilities.

Traceability links could be used to check whether implemented domain artefacts are consistent with the expected enhancements of the smart building. The requirement changes could be easily modelled directly by feature diagrams. Hence, feature choice would directly have an influence on the IT infrastructure. Figure 4 presents an excerpt of a feature diagram describing the kind of functionalities that we can expect from a building equipped with a BMS. *HVAC* and *Lights* features represent systems that can be managed in a building. Their respective sub-features are specific function descriptions representing either monitored data or actuator commands. An expert from the BMS domain could create an exhaustive feature diagram that will be put in relation with a UML component diagram as depicted in Figure 5.

Figure 2 presents high-level features that could be implemented in the project. Each feature should be linked to other ones present in other feature diagrams, such as the one in Figure 4.

4 Gathering Domain and IT Infrastructure Models thanks to Model Driven Engineering

4.1 Software Product Line Application to Domain Models

As said before, smart building IT infrastructure should be able to achieve energetic simulation with monitored data and to optimize machine specific parameters (*e.g.* air stream in HVAC units) to enhance global building energy efficiency.

A *building model* contains, among many others, information like:

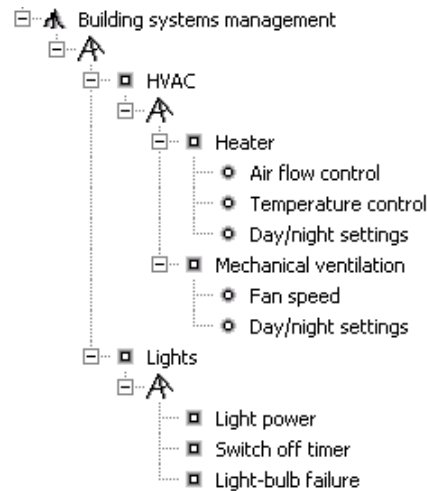


Fig. 4. Excerpt of a features diagram related to buildings domain

- The kind of installed HVAC unit.
- The presence or absence of thermostat in offices.

A *building meta model* contains a repository of the available kinds of components that could be used while modelling a building, including for example:

- List of HVAC parameters
- List of sensor measure types

Figure 5 shows a small excerpt of components that could be instrumented in an office building.

Instrumenting all parameters is very expensive at a large scale. For instance, it might not be always necessary to measure the power of light perceived by human eye in office building rooms. However there might be certain cases when it is required by the client. Figure 6 depicts an excerpt of feature diagram describing what kind of optimisation we can expect from instrumenting a building. *E.g. Room heating control* is something that can be optimized by reducing ventilation noise, if disturbing, or programmed to keep a constant temperature. The idea would be to link features to the modelling artefact necessary to satisfy them. For instance, selecting *room light control* feature would require instrumenting room lights and being able to fill in the parameters:

- *light state* to know whether a light is on or off,
- *light order* to change the light state.

We explained earlier that some features could be expected from the installed IT smart building infrastructure. Therefore, the possible software variations described in *variability diagrams* (as explained in [3]) should help people connecting the building to the RIDER system to check whether the right components

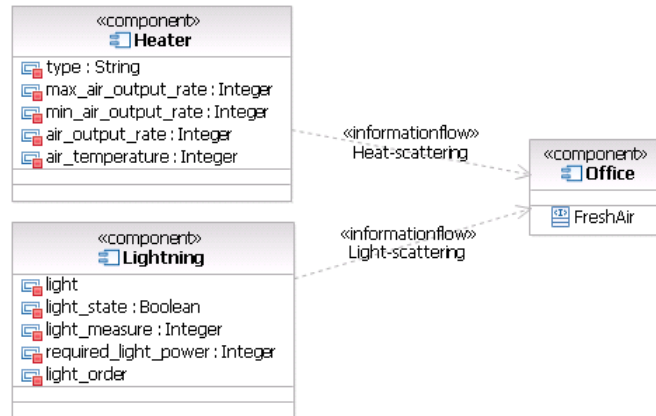


Fig. 5. Excerpt of BMS components diagram

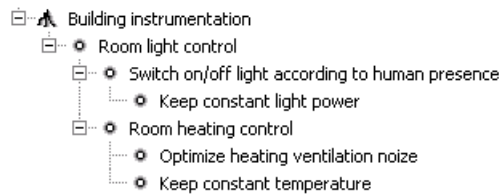


Fig. 6. Some features for instrumenting a building

are instrumented and reciprocally whether some optimization features could be enabled at low cost because of already instrumented components.

If we want to enable the feature *keep constant light power* we shall be able to check that the attribute *light_measure* in the component *Lightning* can be filled by the underlying IT infrastructure.

The next section will present how software product lines could help in creating an IT architecture able to support the diversity outlined before.

4.2 Software Product Line Application to RIDER IT Architecture

One first challenge related to smart buildings is that there exists a very large set of available BMS, sensors, and actuators. BMS manufacturers must be able to easily connect their products to the RIDER system to get the best of RIDER optimization scenarios and analysis. Furthermore, depending on building owner functional and financial requirements, the kind of data to be monitored (*e.g.* CO₂, human presence) or controlled (*e.g.* building mechanical ventilation fan speed) vary widely.

Hence, the RIDER architecture must be very scalable to the size of the building or house, but also adaptable to the nature of optimizations we expect from it and to the level of instrumentation. A base platform, as defined in [11], should be used to allow easy component reuse:

“A platform is any base of technologies on which other technologies or processes are built.” [11]

Such modular needs could be fulfilled by a software platform [12]. A platform could use the Rational Asset Specification (RAS) to ease the reuse of specific developed or configured software applications. For instance, a *database* asset could be reused when the client wants *a posteriori* analysis of measured data of his building. Furthermore, functions of the features he chose to implement and the size of the instrumented building, an estimation of the sizing of the necessary servers could be made. Indeed, traceability through architecture models and feature diagram could help to achieve a sizing of necessary servers and the cost of the IT infrastructure. [13] shows the importance of traceability and gives informations for enabling traceability methods in a project. [14] presents a model-driven component based software product line framework able to automate production of many of product artefacts.

[15] describes how a base framework like the Open Services Gateway initiative (OSGi) can support the variability of the instrumented part of the building. An OSGi bundle could be a specific communication interface between RIDER IT infrastructure and a specific BMS instrumenting a building. Such architecture could greatly improve the openness of the project to other communication and instrumentation standards.

Another problem we will face concerns energy optimization scenarios which can be modelled by business rules. Feature diagrams could also be linked to the elements managed in business rules. We could create feature diagrams describing scenario sets (such as *Building heating optimization* for grouping all kinds of rules related to heating optimization). It could be interesting to suggest to the final user energy-efficiency optimization scenarios according to the existing possibilities of the infrastructure he chose (which depends on his financial and functional requirements).

A similar problem will be faced concerning data analysis. Indeed, some algorithms could be run exclusively when the BMS sends a large enough data flow, or when enough data are at RIDER disposal. Feature diagrams could help targeting the IT requirements of such algorithms. This contribution is also valid for visualization methods.

5 Conclusion and Perspectives

In order to apply software product lines to create control systems for smart buildings, we have presented how we plan to apply and extend existing concepts, like feature diagrams or extensible and component-based architectures, to

this project. Based on these concepts, our additional main challenges are the following:

1. Gathering information from different domain experts to model their fields by linking domain-specific concepts to requirements and IT components.
2. Managing the whole software life-cycle of a smart building related IT project. It includes the following points:
 - Generating domain models from selected features. The generated models would show the minimal building instrumentation requirements to satisfy client expectations.
 - Specifying domain related features (*e.g.* available sensors, actuators, building materials, *etc.*) and generate high-level feature diagrams (for clients) containing only the features that could be implemented without modifying the existing building instrumentation.

The next immediate step will be to synthesize existing feature diagrams semantics, that are the most pertinent for our needs, into a new meta model which could be used as a basis for a plug-in integrated in a modelling tool. This will be used as a basis to enable model transformations based on feature diagrams, UML models and traceability through the different models.

References

1. Stahl, T., Völter, M., Czarnecki, K.: Model-driven software development: technology, engineering, management. John Wiley & Sons (2006)
2. Sendall, S., Kozaczynski, W.: Model transformation: The heart and soul of model-driven software development. *IEEE Software* **20**(5) (2003) 42–45
3. Pohl, K., Böckle, G., van der Linden, F.: *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer (2005)
4. van der Linden, F.J., Schmid, K., Rommes, E.: *Software Product Lines in Action: The best industrial practice in product lines engineering*. Springer Berlin Heidelberg (2007)
5. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute (November 1990)
6. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering* **5**(1) (1998) 143–168
7. Weiss, D.M., Lai, C.T.R.: *Software Product-Line Engineering - A family-based software development process*. Addison-Wesley, Reading, Massachusetts (1999)
8. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice* **10**(1) (2005) 7–29
9. Fey, D., Fajta, R., Boros, A.: Feature modeling: A meta-model to enhance usability and usefulness. *Lecture Notes in Computer Science* **2379** (2002) 198–216
10. Zhang, W., Zhao, H., Mei, H.: A propositional logic-based method for verification of feature models. *Lecture Notes in Computer Science* **3308** (2004) 115–130

Possompès et al.

11. techtarget.com: What is platform? - definition from whatis.com.
http://searchservervirtualization.techtarget.com/sDefinition/0,,sid94_gci212797,00.html (2010)
12. Meyer, M., Lehnerd, A.: The power of product platforms. Free Press, New York (1997)
13. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. *IBM Systems Journal* **45**(3) (2006) 515–526
14. Rubin, J., Yatzkar-Haham, T.: IBM R&D Labs in Israel | consumer electronics development environment (COMPETENCE). <https://www.research.ibm.com/haifa/projects/services/competence/index.shtml> (February 2007)
15. Pham, H.N., Mahmoud, Q.H., Ferworn, A., Sadeghian, A.: Applying Model-Driven development to pervasive system engineering. In: SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments, Washington, DC, USA, IEEE Computer Society (2007) 7