Elaboration on Terms and Techniques for Reuse of Submodels for Task and Workflow Specifications

Peter Forbrig[1(✉)] and Christian Märtin[2]

[1] Department of Computer Science, University of Rostock, Albert-Einstein-Str. 22, 18059 Rostock, Germany
Peter.Forbrig@uni-rostock.de
[2] Faculty of Computer Science, Augsburg University of Applied Sciences, An der Hochschule 1, 86161 Augsburg, Germany
Christian.Maertin@hs-augsburg.de

Abstract. In this paper, terms and techniques are revisited to discuss a terminology for different kinds of reuse. It identifies problems of currently used terminology and suggests using the term generic submodel. An ontology for terms like template, component, subroutine, pattern, and generic submodel is provided.

Keywords: Patterns · Components · Templates · Task models · Business process models · BPMN · Generic components · Patterns · Generic submodel

1 Introduction

Model-based approaches have proven to be useful for developing interactive systems [12]. However, creating models is still challenging. Even with tool support modeling of real world problems is still time consuming an error prone.

The number of errors can be reduced if existing building blocks can be assembled that have proven to be useful in other projects. This phenomenon is well known from programming. The reuse of already tested code fragments is very helpful to improve the quality of software. For programming subroutines, templates, macros, components, and patterns are known concepts to reuse already specified knowledge.

There have been attempts to apply similar ideas to specifications describing the activities of humans in form of task models or workflow specifications. We will have a look at such approaches and try to harmonize the terminology.

The paper is structured in such a way that reuse of task models and reuse of workflow specifications are revisited and discussed separately. Afterwards, it is tried to find a common terminology in a kind of ontology. The paper ends with a summary and an outlook.

2 Reuse of Models

From our point of view, the reuse of models in general is best supported by Design Patterns [11]. They allow the reuse of modeling knowledge on an abstract level.

M. Kurosu (Ed.): HCI 2016, Part I, LNCS 9731, pp. 467–475, 2016.
DOI: 10.1007/978-3-319-39510-4_43

First, a submodel has to be identified that is a candidate for a pattern application. After a pattern was identified that is appropriate, the solution has to be adapted to the context provided by the submodel. An instance of the pattern is created that is ready for integration (application). The four application steps of design patterns can be formalized in the following way.

1. **Identification**: A subset M' of the original model M is selected for pattern application $\mathbf{M'} \subseteq \mathbf{M}$
2. **Selection**: A pattern P is selected that can be usefully applied to M'
3. **Adaptation**: The pattern P is adapted to the current context C of M'. As a result a pattern instance I is delivered. $\mathbf{A}(\mathbf{P}, \mathbf{C}) = \mathbf{I}$
4. **Integration**: The instance I is integrated into M' $\mathbf{F}(\mathbf{M'}, \mathbf{I}) = \mathbf{M'}^{*}$ (which also changes M to M*)

Design patterns are generic solutions for reoccurring problems. UML provides a notation for applying design patterns. Figure 1 provides an example for that.

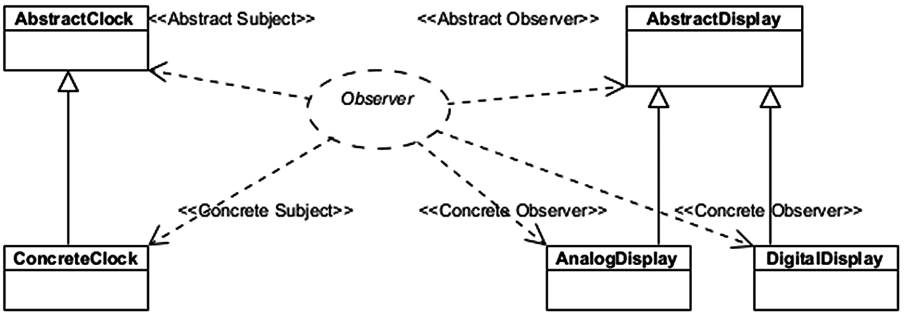

Fig. 1. Application of the GoF patterns observer

Identification was performed by selecting the appropriate classes. Selection was done manually to insert the pattern symbol with the name observer. Adaptation is represented by the connecting arcs. In the above example, there are e.g. two concrete observers. Integration is not represented. The consequences of the pattern application have to be done manually. However, tool support is provided for pattern application in a different way by some case tools.

Additionally to the application of patterns, UML has a notion for generic classes. However, tool support is missing for presenting the resulting models in all case tools we know.

In the following two paragraphs, we will have a look at approaches applying ideas of patterns and genericity for models able to describe activities of humans.

2.1 Reuse for Task Models

In [3] the term task pattern was discussed the first time. However, it was defined as sub-tree only. Within the HAMSTERS environment ([9, 10]) this kind of reuse is called submodel.

The usage of generic task patterns was suggested in [7]. Tool support for task patterns was discussed in [13]. It was shown how generic task patterns were adapted to the context of use and inserted into a larger model. However, in both cases the substitution of parameter values was considered to be performed during design time only. In [5] the runtime substitution of parameter values was discussed the first time for such models. For those patterns, the notation of HAMSTERS was used that allows submodels, procedures, and conditional sub-trees.

To get an impression of the notation, we will specify an example that was given as introduction to BPMN modeling [1]. It originally specifies that it is noticed that somebody recognizes to be hungry. Groceries have to be acquired to be able to prepare the meal. After preparing the meal, it can be eaten. Afterwards, hunger is satisfied. Pre- and post-conditions are not visually represented in Hamsters. Therefore, the task model looks like the graphics presented in Fig. 2.

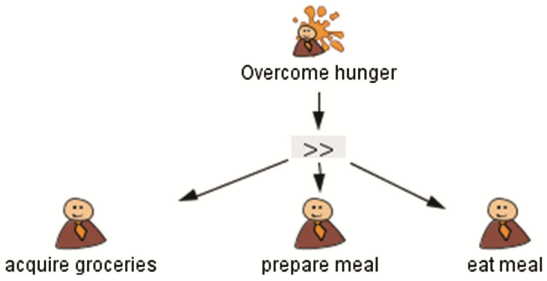

Fig. 2. Task model for overcoming hunger

The presented model can be generalized in such a way that activities of thirsty people can be covered as well. The following generic task model component fulfills the mentioned requirements. It can be instantiated to models for hungry and thirsty people (Fig. 3).

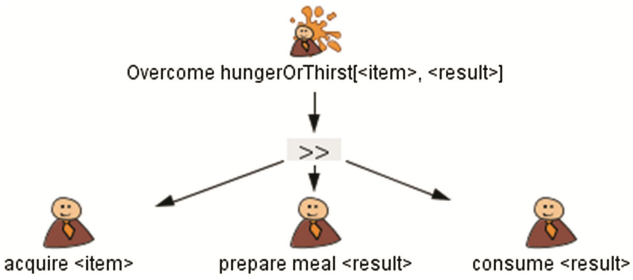

Fig. 3. Generic component to overcome hunger or thirst

Two parameters are used. The first one is called item and can have the values "groceries" for meals and "ingredients" for drinks. The second parameter result can have the values "meal" and "drink". Instantiating the task component with "groceries" and "meal" results nearly in the original model. The only difference is the task eat meal that is generalized to consume meal to abstract eat and drink to consume. A further parameter could have been introduced to deliver exactly the original model.

Additional, to simply propagating values it is possible to make decisions based on these values during design time as well as during runtime. Within runtime, it can be decided, whether values for parameters are used at instantiation time or at decision time.

2.2 Reuse of Workflow Models

Van der Aalst et al. introduced 20 workflow patterns in [15]. Most of these patterns are discussed on a low level of abstraction (see also [16, 17]). They describe solutions for workflow patterns on the level of language features like sequences, alternatives, procedure, metaphors etc.

They are not on the level of describing something like buying or managing something. In [6] we introduced the concept of generic components for BPMN specifications. The hungry people example that was already used for task models [1] was generalized to a generic component. Figure 4 provides this example.

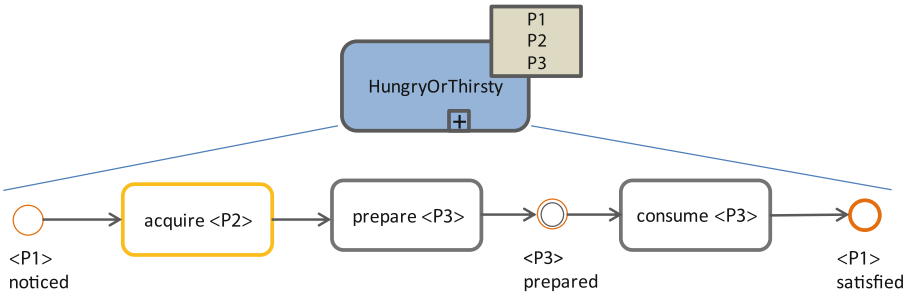

Fig. 4. Generic workflow component for supporting hungry or thirsty people

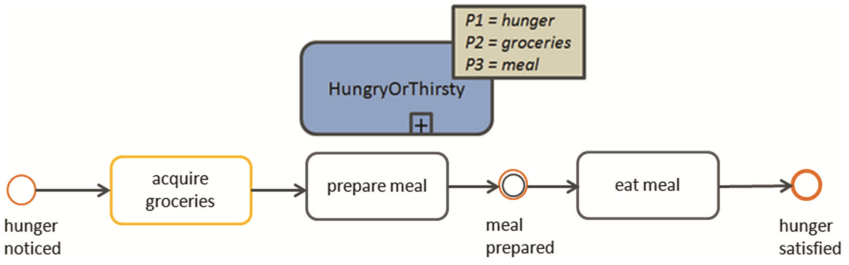

Fig. 5. Instantiation and instance of a generic workflow component

Parameter P1 is expected to have the values “thirsty” or “hungry”. The second parameter P2 specifies the required resources. It can have the values “groceries” or “ingredients”. The third parameter P3 specifies the wanted final result. The values could be “drink“ or “meal”. Figure 5 presents the component with parameter values “hunger”, “groceries”, and “meal” and the corresponding instance of the pattern with substituted parameters.

Hopefully, the selected example allows a good understanding of the idea and a good comparison of task models and workflow specifications. However, it might be too tiny to show the advantages of reuse of existing generic component.

Therefore, an example for ordering a pizza is recalled and generalized. The pizza that is ordered is represented by parameter <sthg> which stays for something. The term “Hungry for Pizza” is generalized to “Keen on <sthg>”. Correspondingly, “Hunger satisfied” is replaced by “Satisfied with <sthg>. Additionally, the roles “pizza customer”, “pizza vendor” and “pizza chief” are replaced by “<sthg> customer”, “<sthg> vendor” and “<sthg> creator” respectively. The resulting generic workflow component is presented in the following Fig. 6.

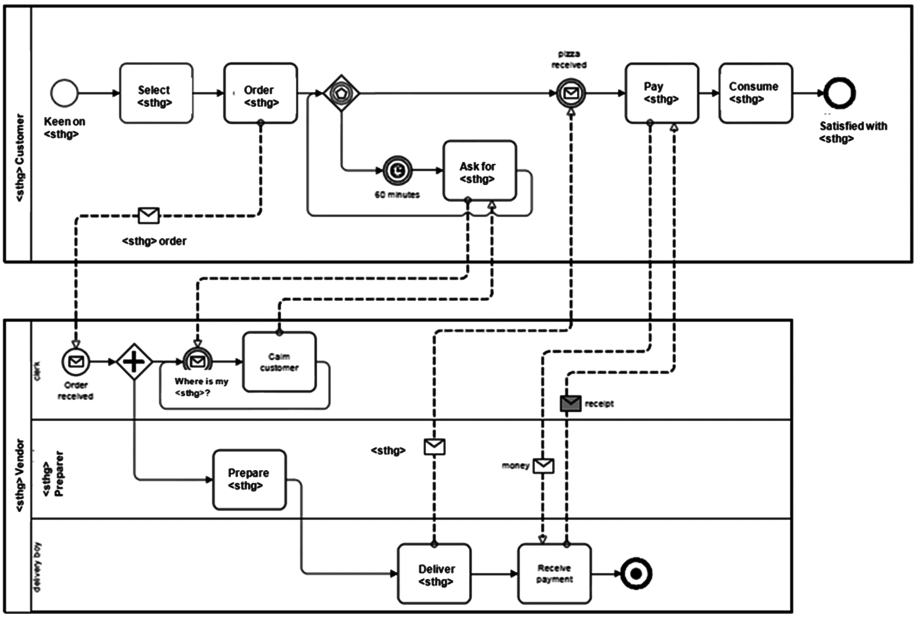

Fig. 6. Generic workflow component based on the pizza ordering example from [2]

The generic workflow component allows not only the specification of ordering pizzas but also of sandwiches in a simple way.

Analysts or business process modelers have simply to instantiate the generic submodel accordingly by providing appropriate parameter values.

2.3 Discussion

Currently, generic components are implemented only for task models for the HAMSTERS case tool [8]. In a case study existing task models describing tasks for a ground segment application was reengineered.

The application is used to monitor and to control the Picard satellite that was launched in 2010 for solar observation. More details about the system are discussed in [5, 9, 10].

The task models became smaller and better readable. The identification of possible component applications allowed even to correct an existing modeling mistake.

Some statistical data are recalled in the following Table 1.

Table 1. Quantitative comparison of the approach (from [5])

Structuring methods	Number of tasks	Number of operators	Reduction percentage
Flat model (without composition mechanisms)	59	13	Reference model for calculus
Structuring with submodels, subroutines and components	35	7	41 % less tasks and 46 % less operators than in flat model

Using structuring mechanisms allowed to reduce the size of the specification by nearly 50 %. This makes us hope that workflow specifications can benefit in the same way. Their similarity was already discussed in [4].

The wording for the different technologies is quite difficult to follow. The term pattern e.g. is used quite differently. Within the context of workflow specifications, it is used quite specific. In the context of workflows, a sequence of actions is called a pattern [15]. This is quite different for task models. In the context of tasks, a pattern is considered to be something more abstract like buying something or running a shop. It is also considered to be generic [7].

While reviewing the models of an existing project we identified our generic task patterns as reusable submodels. They allowed us to model the project in a more compact and better readable way [5]. Our first submission used the term generic patterns.

However, reviewers complained that the identified generic submodels are too specific to be patterns. It was suggested to use the term generic component. Indeed our specified submodels are in no way general patterns that have to be taught to analysts. They are in some way similar to subroutines. In addition to subroutines of Hamsters, task names are not the same all the time. With respect to this aspect, the notion of a template describes the behavior in a better way.

However, templates assume to get all values for generic parameters at the same time. Most of the time this parameter substitution is performed during design time. This could be extended to instantiation time or even runtime. Nevertheless, the combination of value propagation during design time, instantiation time, and runtime is not part of the concept of templates.

Therefore, the term generic component was used in the implementation and in several publications.

Unfortunately, some aspects of the technology we are proposing do not fit to the concept of components. In fact, components support information hiding, which is not true for our ideas. There is one important aspect we want to support. Stakeholders should be able to see the instances of the so called generic component like in Fig. 5. Details of the execution should be visible and not hidden.

Therefore, the term generic submodel seems to be more appropriate than generic component.

Figure 7 provides the suggested ontology for reusable parts of models

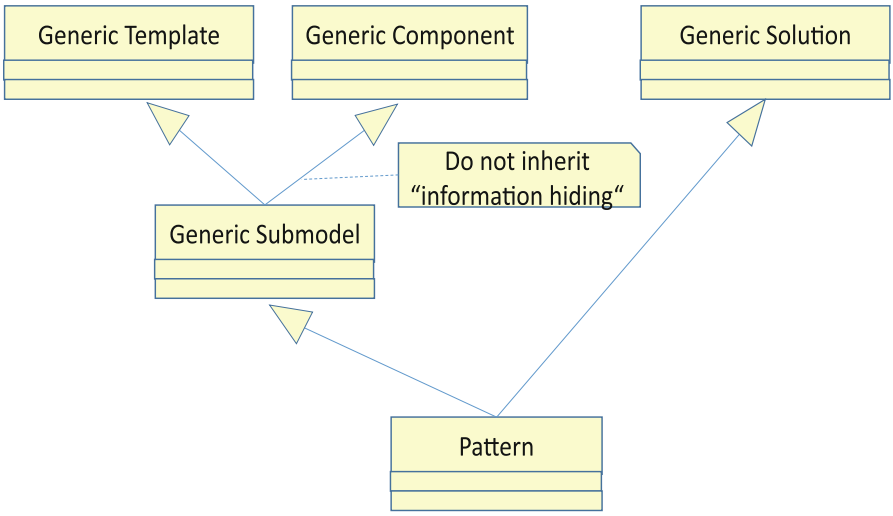

Fig. 7. Suggested ontology

From our point of view, generic submodels inherit all features from templates. Parameters can be used to specify any generic aspect. In this way, names of model elements can be adapted. The same is true for conditions or any expressions.

In addition, generic submodels inherit some features from components. Parameters can be instantiated at several stages of development and execution. However, information hiding is not inherited.

Generic submodels can be patterns. However, not all submodels are patterns. It depends on the level of abstraction and on the importance for general applications. There can be patterns for general solutions and domain-specific patterns that are important for reuse in a specific domain.

Additionally, there might be submodels that are important in a specific project. However, these submodels are not general or abstract enough to be considered as patterns. This is reflected by the inheritance relation from generic submodels to patterns. Specific submodels are patterns but not all of them.

However, patterns do not have to be submodels in the sense of templates and components. They can provide general ideas as well.

This reflected in our suggested ontology by providing inheritance from class generic solution to class pattern.

3 Summary and Outlook

This paper has discussed some strategies of reusing models. It started with the application of design patterns and generic classes for object-oriented class diagrams and continued with task models and workflow specifications describing activities of users. Such specifications are very important for human-centered design of interactive systems in general and for user interfaces in detail.

It was discussed, how terms like templates, components, submodels, and patterns are related to each other. It was suggested to use the term generic submodel in connection with tool support for task models because not all useful model fragments are patterns. Additionally, patterns do not have to be some kind of template, component, or submodel. They can provide more general and abstract ideas as well.

Additionally, it is important to notice, that users can be presented models with appropriate names and terms from the domain, that is modelled and that is familiar to the users. Names and terms are presented in a way they are used to. Models can be easily reused in a way that allows user friendly specifications.

Based on the experience with task models in Hamsters, an implementation of generic submodels is under development for a BPMN editor of an open source projects. Experiments with business process modelers have to show how predefined generic submodels are used.

Later, libraries of already specified submodels have to be created that support a good overview of the generic submodels. Additionally, they have to provide efficient search strategies.

References

1. BPMN: http://www.bpmn.org. Accessed 03 Feb 2016
2. BPMN 2.0 Tutorial: https://camunda.org/bpmn/tutorial/#simple_flow_BPMN. Accessed 03 Feb 2016
3. Breedvelt-Schouten, M., Paternò, F., Severijns, C.: Reusable structures in task models. In: Harrison, M.D., Torres, J.C. (eds.) Proceedings of DSV-IS 1997, pp. 225–239. Springer, Heidelberg (1997)
4. Brüning, J., Dittmar, A., Forbrig, P., Reichart, D.: Getting SW engineers on board: task modelling with activity diagrams. In: Gulliksen, J., Harning, M.B., Palanque, P., van der Veer, G.C., Wesson, J. (eds.) EIS 2007. LNCS, vol. 4940, pp. 175–192. Springer, Heidelberg (2008)
5. Forbrig, P., Martinie, C., Palanque, P., Winckler, M., Fahssi, R.: Rapid task-models development using sub-models, sub-routines and generic components. In: Sauer, S., Bogdan, C., Forbrig, P., Bernhaupt, R., Winckler, M. (eds.) HCSE 2014. LNCS, vol. 8742, pp. 144–163. Springer, Heidelberg (2014)

6. Forbrig, P.: Generic components for BPMN specifications. In: Johansson, B., Andersson, B., Holmberg, N. (eds.) BIR 2014. LNBIP, vol. 194, pp. 202–216. Springer, Heidelberg (2014)
7. Gaffar, A., Sinnig, D., Seffah, A., Forbrig, P.: Modeling patterns for task models. In: Proceedings of the 3rd Annual Conference on Task Models and Diagrams (TAMODIA 2004), pp. 99–104. ACM, New York (2004)
8. HAMSTERS: http://www.irit.fr/ICS/hamsters/
9. Martinie, C., Palanque, P., Winckler, M.: Structuring and composition mechanisms to address scalability issues in task models. In: Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., Winckler, M. (eds.) INTERACT 2011, Part III. LNCS, vol. 6948, pp. 589–609. Springer, Heidelberg (2011)
10. Martinie, C., Palanque, P., Ragosta, M., Fahssi, R.: Extending procedural task models by explicit and systematic integration of objects, knowledge and information. In: Proceedings of ECCE 2013, article no. 23, pp. 1–10 (2013)
11. Gamma, E., et al.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston (1995)
12. Kruschitz, C., Hitz, M.: Human-computer interaction design patterns: structure, methods, and tools. Int. J. Adv. Softw. **3**(1 & 2) (2010)
13. Radeke, F., Forbrig, P.: Patterns in task-based modeling of user interfaces. In: Winckler, M., Johnson, H. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 184–197. Springer, Heidelberg (2007)
14. Sinnig, D., Gaffar, A., Reichart, D., Seffah, A., Forbrig, P.: Patterns in model-based engineering. In: Jacob, R.J.K., Limbourg, Q., Vanderdonckt, J. (eds.) Proceedings of CADUI 2004, pp. 197–210. Springer, Heidelberg (2004)
15. Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(3), 5–51 (2003)
16. White, S.A.: Process modeling notations and workflow patterns. BPTrends (2004). http://www.omg.org/bp-corner/bp-files/Process_Modeling_Notations.pdf. Accessed 22 Nov 2015
17. Wohed, P., van der Aalst, W.M., Dumas, M., ter Hofstede, A.H., Russell, N.: On the suitability of BPMN for business process modelling. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 161–176. Springer, Heidelberg (2006)