Process-Integrated Refinement Patterns in UML

Timo Kehrer
Dept. of Computer Science and Media
Stuttgart Media University (HdM)
Nobelstr. 10, D-70569 Stuttgart, Germany
Tel: +49 711 8923 2619 –Fax: +49 711 8923 2188 – e-mail: kehrer@mi.hdm-stuttgart.de

Edmund Ihler
Dept. of Computer Science and Media
Stuttgart Media University (HdM)
Nobelstr. 10, D-70569 Stuttgart, Germany
Tel: +49 711 8923 2166 –Fax: +49 711 8923 2188 – e-mail: ihler@hdm-stuttgart.de

Abstract: The Unified Modeling Language (UML) is a widely used standard in Model-Driven Engineering (MDE). Using the UML in a software development process means to refine and evolve models in many ways. Firstly, a system model evolves through different layers of abstraction towards an appropriate design in an object-oriented programming language (vertical refinement). Secondly, a set of consecutive revisions is produced within a level (horizontal refinement). Whereas the UML supports the specification of a system at all levels of abstraction, the concept of refinement lacks precise semantics and is open to misconceptions. As a general-purpose modeling language, there are no precise conceptual guidelines on how to use the wide range of UML diagrams in a development process. The semantics of a specific kind of refinement most often requires the context, i.e., the triggering development activity of the enacted process model, to be taken into account. Refinement relationships have to be documented manually, which is a very error-prone and tedious work. In this position paper, we outline our ongoing work on developing a framework for the specification and operationalization of UML refinement patterns.

Key words: UML, MDE, refinement patterns, model modification templates, process meta-model, automated traceability

1. INTRODUCTION

In Model-Driven Engineering (MDE), the Unified Modeling Language (UML) [21][22] became a widely used standard in modeling object-oriented software systems. Using the UML in a software development process means to refine and evolve models in many ways. Firstly, a system model evolves through different layers of abstraction towards an appropriate design in an object-oriented programming language (vertical refinement). Secondly, a set of consecutive revisions is produced within a level (horizontal refinement). Refinement is one of the cornerstones of a formal approach to software engineering. Its traditional purpose is to show that an implementation or concrete specification meets the requirements of an abstract specification. The basic idea is that an abstract specification can be substituted by a concrete one as long as its behavior is consistent with that defined in the abstract specification. Stepwise refinement allows this process to be done in stages [2], a powerful refinement machinery is present in most formal languages. Whereas the UML supports the specification of a system at all levels of abstraction, and the underlying meta-model allows using and processing model information in a well defined way, the concept of refinement lacks precise semantics and is open to misconceptions. A number of different alternatives to a more formal notion of refinement for the UML have been proposed.

One strategy is to translate a UML model into a formal modeling notation. Thus, the semantic relation between the model elements can be studied by means of their images in the semantic domain. A number of different mapping approaches and consistency checking strategies have been proposed. The approaches presented in [19], [12] and [23] belong to this category. Further, as UML models can be interpreted as graphs, another strategy towards a formalization of UML refinements consists in applying the theory of graph transformation. The works presented in [10] and [7] are related to this category. The approaches of both categories are appropriate to discover and correct inconsistencies and ambiguities. They allow the verification of refinements for a restricted set of UML modeling constructs. However, the proposed approaches are non-constructive, i.e., they provide no feedback in terms of the UML [18].

Additionally, a number of lightweight approaches to a formalization of UML refinements have been proposed. Paige et al. [16] define refinement in terms of model consistency. Consistency rules, expressed in the Object Constraint Language (OCL) [15], are used to provide a formal definition of refinement for meta-models conforming to the Meta-Object Facility (MOF) specification [14]. Lano et al. [13] describe some patterns for the refinement of UML specifications into executable implementations, using a semantically precise subset of the UML named UML-RSDS [11]. A specific development process supported by UML-RSDS as well as an accompanying toolset is provided. Pons et al. [17][18] adopt a set of well-founded refinement structures provided by Object-Z [1] to define UML refinement structures. They use OCL constraints to express the respective retrieve relations and refinement conditions. The Catalysis Method [3] provides traceability by documenting retrieve relations using UML associations.

Lightweight approaches are more suitable to industrial software development as they are based on languages, tools and methods that are widely accepted and understood in the MDE community. However, in many cases, many constraints have to be specified in order to document refinement relations in OCL. This is very error-prone and tedious work. Moreover, as a general-purpose modeling language, the UML lacks precise conceptual guidelines on how to use the wide range of diagrams in a development process. The semantics of refinements most often require the context, i.e., the triggering development activity of the enacted process model, to be taken into account. Furthermore, most of the presented lightweight approaches only consider a semantically precise subset of the UML. Thus, the full potential of the UML is not tapped.

In this position paper, we outline our ongoing work on developing a framework for the specification and operationalization of UML refinement patterns. The application of refinement patterns is considered as software development activity integrated into a process model. Documentation of refinement relationships is based on the UML abstraction mechanism which will be briefly introduced in Section 2. We extend the mechanism in several ways. Section 3 presents our approach of formally integrating refinements into a software development process. Section 4 introduces our approach to model modification through UML refinement patterns that is based on the template-mechanism of the UML. Section 5 concludes the paper and states some research questions and hypotheses that will be investigated by our future work.

2. REFINEMENT IN TERMS OF THE UML

Refinement relations in the UML are modeled using abstraction relationships. In the UML meta-model [1], *Abstraction* is a special *Dependency* in which there is a mapping between a client (or clients) and a supplier (or suppliers) representing the same concept at different levels of abstraction or from differing viewpoints. A mapping specification allows the specification of the relationship in a formal way, as for example, using the OCL (cf. Fig. 1).

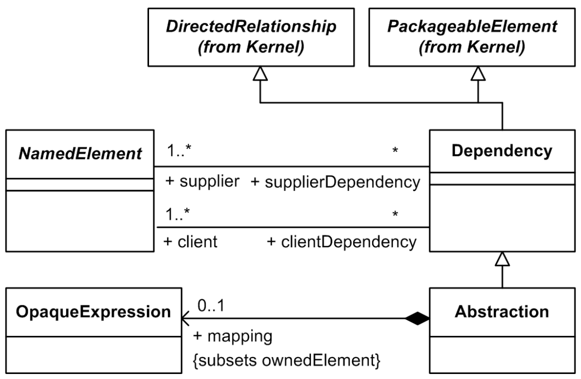

Fig. 1. Refinements in terms of the UML

A set of predefined stereotypes can be applied to abstraction relationships. The stereotype «refine» specifies a refinement relationship between model elements at different levels of abstraction, such as analysis and design. The stereotype «trace» specifies a relationship between model elements or sets of model elements that represent the same concept in different models. Traces are mainly used for tracking requirements and changes across models. The UML does not specify any further semantics of refinement. Most often, the concrete semantics result from the constituting development activity of the enacted process. A formal approach to the integration of refinements into structured process models is presented in the following section.

3. PROCESS INTEGRATION OF REFINEMENTS

We consider process models in a structured manner adopting the Software & Systems Process Engineering Meta-Model Specification (SPEM) proposed by the Object Management Group (OMG) [20]. More precisely, we adopted the Unified Method Architecture (UMA) which is the underlying meta-model of the Eclipse Process Framework (EPF) [4]. As major parts of the UMA went into revision 2.0 of SPEM, the concepts introduced by SPEM and UMA can be basically considered as equivalent. Section 3-1 introduces the most relevant SPEM concepts with respect to the integration of refinements. The according extensions to SPEM will be presented in Section 3-2.

3-1 Basic SPEM concepts

SPEM is a process engineering meta-model as well as a conceptual framework providing concepts for modeling development methods and processes. On the one hand, SPEM supports the standardization and management of libraries of reusable methods and key practices (method content). On the other hand, it supports the development of appropriate process models (process structure). In the meta-model, the distinction between method content and process structure is reflected by two disjoined inheritance hierarchies introducing the respective meta-classes. Figure 2 shows a subset of SPEM process structure (left) and method content (right) meta-classes.

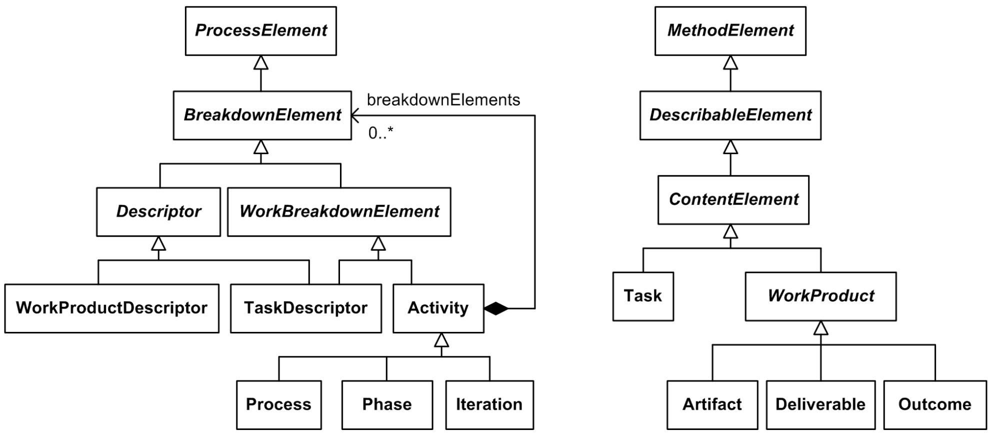

Fig. 2. A subset of SPEM process structure (left) and method content (right) meta-classes

Method contents are integrated into a process structure via descriptor instances. The most relevant concepts with respect to our extensions introduced in Section 3-2 are depicted in Figure 3. A *Task* is informally defined as an assignable unit of work, which contains a complete step-by-step explanation of doing all the work that needs to be done to achieve a specific goal. A task can be invoked many times throughout a development process. Each invocation is defined by an individual *TaskDescriptor*, which is informally defined as a proxy for a task in the context of one specific development activity. Each task descriptor can manage its own invocation specifics with respect to input and output work products that are referenced through a *WorkProductDescriptor*. A *WorkProduct* is informally defined as a tangible artifact consumed, produced, or modified by tasks.

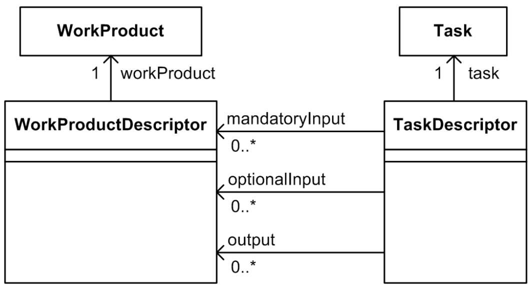

Fig. 3. Integration of method content and process structure in SPEM (clipping)

3-2 Extensions to SPEM

SPEM allows the documentation of development methods and activities in a structured manner. However, the tasks that have to be performed in an enacted process are informally described using natural language. We customized SPEM to assist MDE with the UML in a more formal way. This section introduces the extensions to the meta-model.

First, as models play a key role in MDE, we introduce this specific kind of work product on the SPEM meta-model level (cf. Figure 4, left). Different kinds of models, e.g., Domain Object Model, Analysis Model or Design Model, in terms of the Rational Unified Process (RUP) [9], are defined through the instantiation of the meta-class *Model*. A model is represented by a UML model instance (*UML::Model*). Additionally, an ordered set of consecutive revisions of type *Version* is associated with a specific model instance. A dedicated set of UML model elements (*UML::Element*) associated with a revision represents a view of the model at a specific point of time. Furthermore, we introduce the concept of refinements on the meta-model level defining the meta-class *RefinementPattern* as a special task (cf. Figure 4, right). Thus, refinement patterns can be specified independently of the development activities to which they will be applied. Although UML-collaborations are a more general concept with which to express patterns in UML, we decided to keep the concept overhead as small as possible. Therefore, we use UML templates, more precisely, templated packages, which can be used to generate other model elements by the instantiation of template binding relationships. A template precisely specifies the refinement pattern (cf. Section 4).

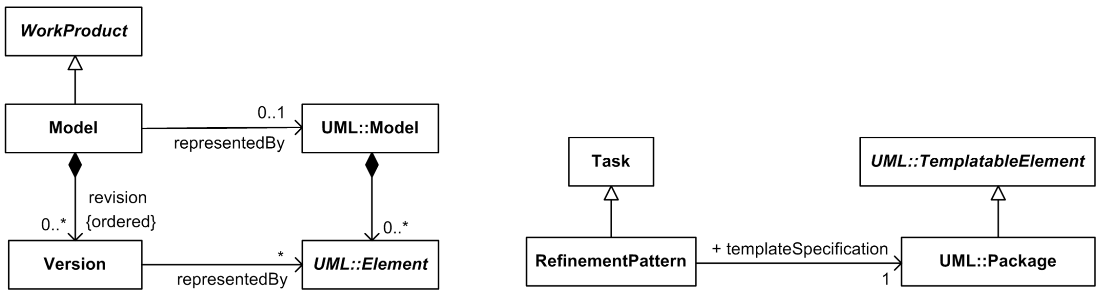

Fig. 4. SPEM extensions (1) and (2): Model (left) and Refinement patterns (right)

Figure 5 shows how the SPEM descriptor concept (cf. Section 3-1) is extended in order to integrate refinement patterns as special development tasks into the process model. Applicable refinement patterns in terms of a process model are referred to as *RefinementOperation,* which reference their respective pattern specifications. A special work product descriptor named *ModelDescriptor* serves as proxy for concrete model instances and declares them as source or target models in terms of a specific refinement operation. Thus, refinement operations can be applied to all kinds of models by any development activity of a process specification. The operationalization of refinement patterns will be described in more detail in Section 4-3.

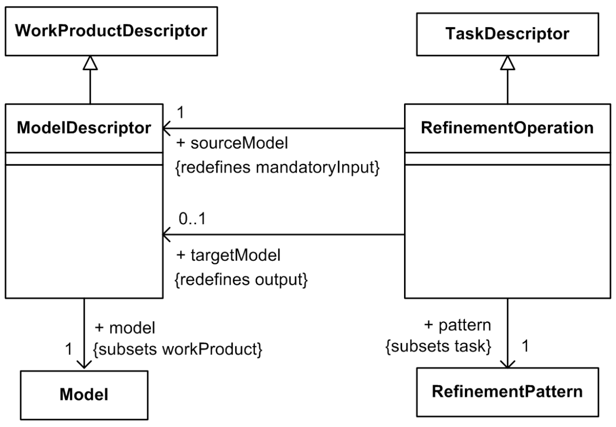

Fig. 5. SPEM extensions (3): Refinement operations

4. TEMPLATE-BASED REFINEMENT OPERATIONS

We adopted the UML template-mechanism in order to define refinement patterns in a precise and intuitive manner. Section 4-1 briefly introduces the UML template-mechanism. Section 4-2 introduces our approach of defining refinement patterns before Section 4-3 finally considers their operationalization.

4-1 The UML Template-mechanism

Figure 6 shows a simplified subset of the relevant part of the UML meta-model [1] with respect to template specifications. A templateable element (*TemplateableElement*) may contain a template signature (*TemplateSignature*) that specifies an ordered set of formal template parameters (*TemplateParameter*). Each template parameter is a kind of pointer referencing a parameterable element (*ParameterableElement*), which is finally declared as a formal template parameter. A classifier template parameter (*ClassifierTemplateParameter*) is a special template parameter that declares a classifier as formal template parameter. It constrains the arguments that may be specified for this template parameter in a binding of the template (cf. Figure 6, left). In the context of template bindings, a templateable element serves as source element of a template binding (*TemplateBinding*), which is a directed relationship. The target element of the binding relationship is the template signature whose parameters are bound. A substitution (*TemplateParameterSubstitution*) relates actual parameter(s), which are parameterable elements, to a formal template parameter defined by the template signature (cf. Figure 6, right).

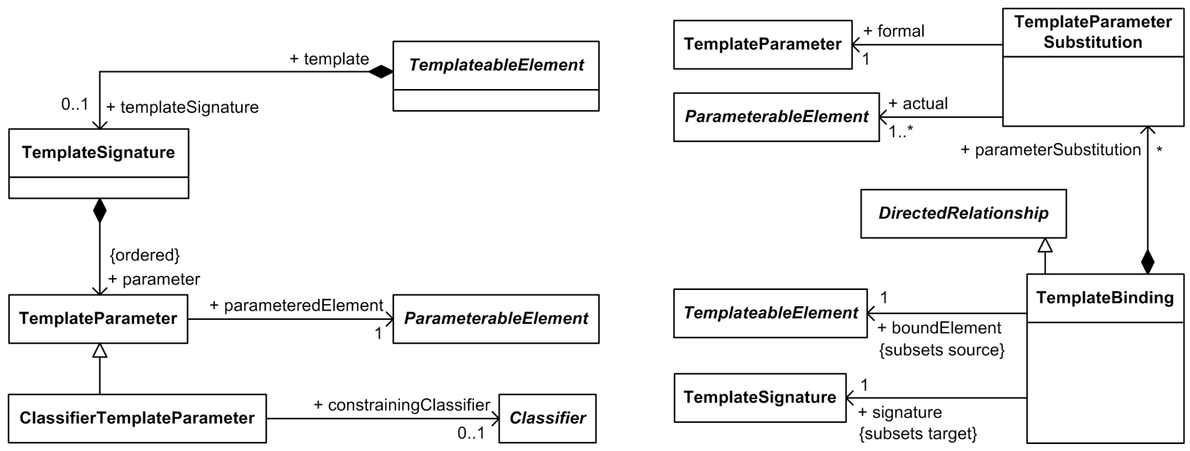

Fig. 6. UML templates (left), UML template bindings (right)

4-2 Specification of Refinement Patterns

Refinement patterns are specified as UML templates. The template signature defining the formal parameters is provided by a UML package (which is a templateable element), in the following, denoted as template package (cf. Section 3-2). Beyond providing a container for the model elements involved in a template specification, a template package has no further meaning, i.e., the package itself does not occur in the model to which the template is applied. We illustrate the concept of template-based refinements by means of a set of examples.

The refinement patterns depicted in Figures 7 and 8 address typical refinements from requirements to analysis, in this case, the mapping of use cases onto a suitable component architecture. Here, a direct mapping of use cases onto (functional) components is specified. The name of the component onto which a use case is mapped equals the name of the use case. The notation <*expr*> is used as a simple string expression which allows addressing the values of meta-attributes of model elements related to the pattern specification. Whereas refinement pattern *uc-2-comp* (cf. Figure 7) addresses a single use case without considering its context, *uc-2-comp-include* (cf. Figure 8, left) and *uc-2-comp-extend* (cf. Figure 8, right) specify how to model «include» and «extend» relationships on the component level by importing and exporting component interfaces. Here, mapping relations of use cases and components are specified as trace links (cf. Section 2), whereas imported and exported component interfaces are considered as refinements (cf. Section 2) of the according use case relationships.

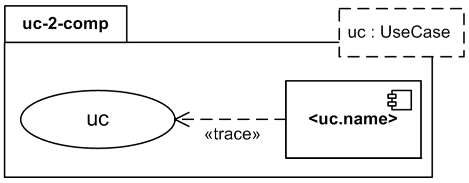

Fig. 7. Mapping of single use cases onto components

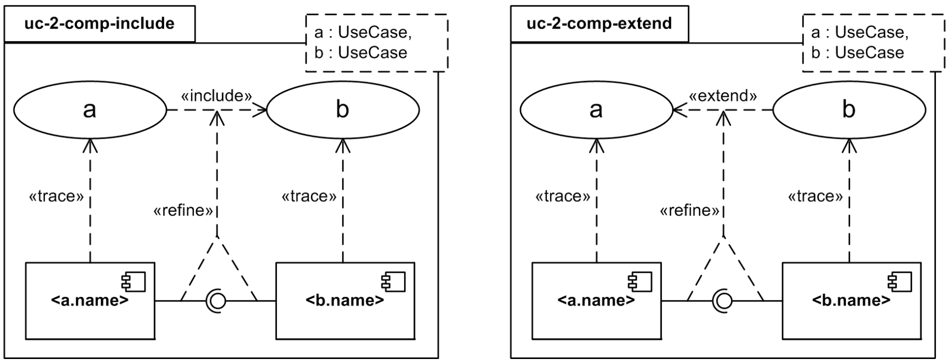

Fig. 8. Mapping of included use cases (left) and use case extensions (right) onto components

A typical refinement activity from analysis to design is to derive a design model suitable for the implementation in an object-oriented language. Typical examples for such constructs are the elimination of association classes (which cannot be expressed in any mainstream object-oriented programming language), or the elimination of many-to-many associations. Figure 9 (left) depicts a possible implementation pattern for associations to 0..* in Java (*java-res-a-n*). In addition to the formal parameters *ass-a-b*, *A* and *B*, the desired collection class (*C*) has to be selected from the Java profile introducing Java specific language constructs into the UML.

An example of design model refinement is addressed by the pattern depicted in Figure 9 (right). With respect to the implementation of associations, a set of canonical navigation patterns can be deduced from association attributes such as navigability and multiplicity. The refinement pattern *nvp-a-0-1* presents, as an example, a set of suitable navigation methods for a navigable association to 0..1.

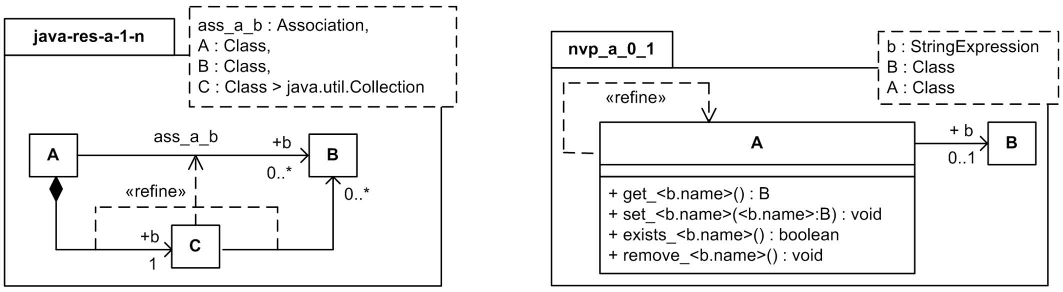

Fig. 9. Resolution of associations to 0..* in Java (left), navigation pattern for associations to 0..1 (right)

4-3 Operationalization of Refinement Patterns

Applicable refinement patterns integrated into the process model are referred to as refinement operations (cf. Section 3-2). They are used to accomplish pattern-based model modifications. A template specification is operationalized to generate other model elements by the instantiation of template binding relationships. Model modifications (or transformations) based on template-bindings can be compared to graph rewriting rules known from graph grammars [8]. The left-hand-side, i.e., the matching part of a rewriting rule, is represented by the selection of the actual parameters. The right-hand-side of a rewriting rule is represented by the template specification.

Figure 10 illustrates the approach by applying the refinement pattern *java-res-1-n* (cf. Section 4-2) to a sample design model, which simply consists of two design classes *Employer* and *Employee*. An employee can be employed by exactly one employer, whereas an employer may have an arbitrary number of employees. That is, employer and employee are in a 0..*-association. According to a (fictitious) process model, 0..*-associations have to be resolved towards an appropriate Java implementation. Applying the refinement pattern *java-res-1-n*, we have to select appropriate bindings for the formal parameters *ass-a-b*, *A*, *B* and the desired collection class (*C*) from the Java profile. In this case, we chose *ArrayList* as appropriate Java collection class.

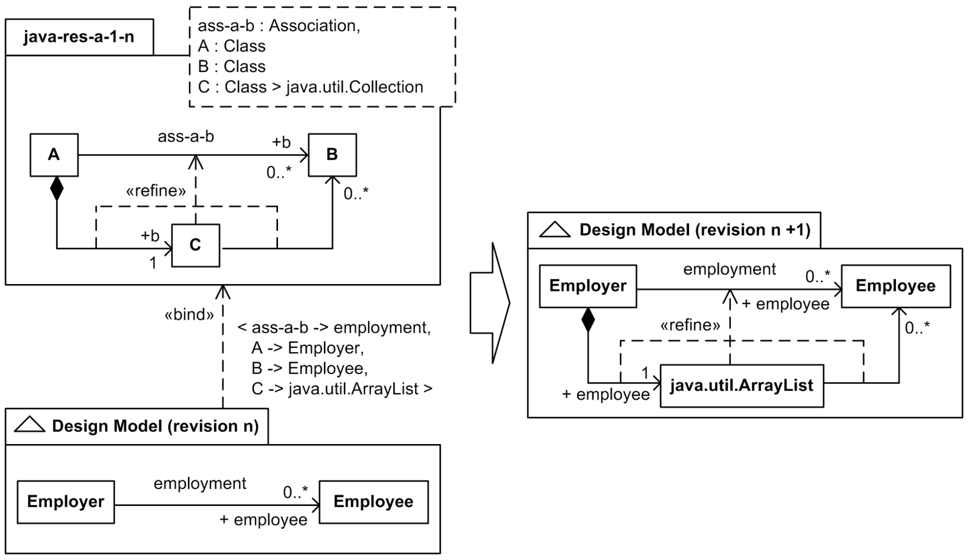

Fig. 10. Application of refinement pattern *java-res-1-n* onto a sample design model

5 CONCLUSION AND OUTLOOK

Using the UML itself to document refinement relationships, we presented an intuitive and precise means of specifying generic refinement patterns for UML models. We adopted the UML template-mechanism in order to define refinement patterns directly in UML. We used the concept of parameter binding to operationalize refinement patterns. A set of extensions to SPEM allows us to integrate applicable refinement patterns, referred to as refinement operations, into the process model and to use them to accomplish pattern-based model modifications. The semantics of refinements are manifested thereby in the underlying process model, and refinements are precisely documented. In addition to documenting the mapping between refined model elements and their abstract counterparts we explicitly retain the links to the development activities which trigger the refinement operations. Thus, strict process conformance and automated traceability are provided. The approach can also be used to precisely document and operationalize design patterns [6] and refactoring operations [5].

A set of various research questions arising from our approach must be investigated. As our template-based approach to model modifications (and transformations) can be compared to graph rewriting rules, we will investigate to what extent the selection of refinement patterns in terms of a process model, i.e., the restriction of the amount of possible model modifications to a finite set of applicable editing operations, can be compared to the definition of a graph grammar consisting of a finite set of graph rewriting rules.

Further, the selected refinement patterns in terms of a process model determine the characteristics of the models that can be produced. We will have to investigate which kind of models can be created by the adoption of which refinement patterns. It will have to be investigated whether a set of refinement patterns is complete in the sense that it can be used to derive the design of a system, or at least a certain group of systems offering some dedicated characteristics. Such a set of refinement patterns is comparable to a Domain Specific Language (DSL) but offers automated traceability and strict process conformance. The question arises whether we can create appropriate sets which cover some relevant domains and speed up the development. In a first step, we will investigate how our framework can be used to adopt other refinement pattern catalogues presented in the literature.

Finally, the industrial applicability of our approach will have to be evaluated by means of a prototypical implementation of the presented concept. This will test whether the definition of refinement patterns as special software development activities will meet the requirements of the MDE community.

REFERENCES

[1] Derrick, J., Boiten, E., Refinement in Z and Object-Z - Foundation and Advanced Applications. Springer (2001)
[2] Dijkstra, E.W., A Discipline of Programming. Prentice Hall Series in Automatic Computation (1976)
[3] D'Souza, D., Wills, A., Objects, Components and Frameworks with UML, Addison-Wesley (1999)
[4] Eclipse Process Framework Project, http://www.eclipse.org/epf/ (2008)
[5] Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., Refactoring - Improving the Design of Existing Code. Addison-Wesley (1999)
[6] Gamma, E., Helm, R., Johnson, R.E., Vlissides, J., Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
[7] Giese, H., Glesner, S., Leitner, J., Schäfer, W., Wagner, R., Towards Verified Model Transformations. MoDELS'06 Workshop MoDeVa, Genova, Italy (2006)
[8] Königs, A., Model Transformation with Triple Graph Grammars. MoDELS'05 Workshop Model Transformations in Practice, Montego Bay, Jamaica (2005)
[9] Kruchten, P., The Rational Unified Process - An Introduction. Addison-Wesley (2000)
[10] Küster, J.M., Definition and Validation of Model Transformations. In Journal on Software and Systems Modeling, 5(3):233-259, Springer (2006)
[11] Lano, K., Clark, D., Androutsopoulos, K., RSDS - a subset of UML with Precise Semantics, L'Objet, 9(4):53-73 (2003)
[12] Lano, K., Clark, D., Androutsopoulos, K., UML to B: Formal Verification of Object-Oriented Models. IFM'04, Canterbury, England (2004)
[13] Lano, K., Androutsopolous, K., Clark, D., Refinement Patterns for UML. REFINE'2005, Guildford, England (2005)
[14] MOF 2.0. Meta-Object Facility Core Specification version 2.0 - OMG Final Adopted Specification (2006)
[15] OCL 2.0. Object Constraint Language Specification version 2.0 - OMG Final Adopted Specification (2006)
[16] Paige, R., Kolovos D., Polack, F., Refinement via Consistency Checking in MDD. REFINE'2005, Guildford, England (2005)
[17] Pons, C., On the definition of UML refinements. MoDELS'05 Workshop MoDeVa, Montego Bay, Jamaica (2005)
[18] Pons, C., Garcia, D., A Lightweight Approach for the Semantic Validation of Model Refinements. In Electronic Notes in Theoretical Computer Science (ENTCS), Elsevier (2008)
[19] Roe D., Broda, K., Russo A., Mapping uml models incorporating ocl constraints into object-z. Technical report, Dept. of Computer Science, Imperial College London (2003)
[20] SPEM 2.0. Software & Systems Process Engineering Meta-Model Specification version 2.0 - OMG Final Adopted Specification (2008)
[21] UML 2.1.2. The Unified Modeling Language Infrastructure version 2.1.2 - OMG Final Adopted Specification (2007)
[22] UML 2.1.2. The Unified Modeling Language Superstructure version 2.1.2 - OMG Final Adopted Specification (2007)
[23] Van der Straeten, R., Mens, T., Simmonds, J., Jonckers, V., Using description logic to maintain consistency between UML-models. UML'03, San Francisco, USA (2003)