# Transformations in Pattern-Based System Specifications

Christian Peper

SFB 501[1]
University of Kaiserslautern
*peper@informatik.uni-kl.de*

## Abstract

The known pattern-based software development approaches concentrate on the discovery, definition and reuse of patterns to improve specification *products*. By contrast, *transformations* can provide valuable knowledge for the system development *process*. The concept of transformations and their relation to patterns are introduced and illustrated.

*Keywords:* patterns, transformations, requirements engineering, reuse, formal specification, temporal logic, real time

## 1 Introduction

The growing interest in reuse and quality management in (software) system development processes has led to a number of pattern-based approaches to the specification of various aspects of a system. Gamma et al. have introduced design patterns in object-oriented software [Ga+94] that have been generalized for SDL [GeGoRö97]. Patterns are also applicable in the area of requirement specifications [PeGoKr97, GoKrPe98]. *Patterns* are certain parameterized specification fragments with well-defined properties. They have the potential to improve the quality of specifications and to increase the controlled reuse of specification parts.

If a pattern-based technique yields executable specifications, the informations about the pattern usage are often discarded. For example, no currently available SDL code generator profits from such pattern instantiation informations. On the other hand, the pattern usage may contain valuable hints for the further implementation. Obviously, this becomes more important at early stages of the system development, where specifications are not executable. The multiple instantiation of the same pattern indicates similar problems, that often have similar solutions (implementations). Relating specification patterns and corresponding solutions can provide valuable design suggestions for the system developer. A *transformation* is such a relation.

The following considerations shall give an impression of the notions *pattern* and *transformation.* These are illustrated based on the *Formal Requirement Specification Technique (FOREST)* [KrPe99c]. The *FOREST* approach relies on a real-time temporal logic to support a property-oriented specification style. This logic has been tailored for the domain of building automation systems. Furthermore, it has been extended by object-oriented abstraction and structuring concepts, such as aggregation, parameterization and inheritance. The resulting formal description technique (FDT) was well suited for the definition and reuse of patterns.

1. This work was supported by the *Deutsche Forschungsgemeinschaft* (DFG) as part of the *Sonderforschungsbereich* (SFB) 501: "Development of Large Systems with Generic Methods".

## 2 Patterns and Transformations

The general definition of the notions pattern and transformation for a greater class of specification techniques is quite complex. Therefore, we can give here only an impression of the first ideas how to formulate such a framework for pattern-based specifications.

To support the definitions, a (formal) *specification technique* should provide:

- a set $S$ of correct specifications (the *specification language*) and a set of *specification fragments* or partial specifications, that can be extended to correct specifications. E.g., $\Box(A \rightarrow \Diamond B)$ is a specification, $\Box(x)$ is a specification fragment, where the variable $x$ represents another specification fragment.
- the definition of the semantics, typically given as a function *sem* that maps each correct specification $s \in S$ to a set $M$ of models that satisfy this specification. The semantics should also provide an equivalence relation "=" and an implication relation "$\subseteq$".

The task of a pattern is to enable the specification of a certain aspect, for which a contiguous specification can not be given, i.e., the specification fragments relevant for this aspect may be spread over the whole specification. In a pattern definition it is important to distinguish between *pattern specification fragments* and *context specification fragments*. Pattern fragments are added to the resulting specification by the instantiation of this pattern. Context fragments are added by anyone else, but they allow to restrict the context in which the pattern instance may be embedded.

A *pattern* consists of finite descriptions for:

- the pattern specification fragments $P$ of the form $p(x_1, ..., x_n)$
- the context specification fragments $C$ of the form $c(y_1, ..., y_m)$
- a composition function *comp:* $X \rightarrow$ C, and $Y \rightarrow P$

Each allowed composition of pattern fragments $p \in P$ and context fragments $c \in C$ must be a correct specification. For example: $B(\Box(A \rightarrow B))$ is an allowed composition and a correct specification for:

$$P = \{\Box(x)\}$$

$$C = \{A, A \rightarrow B, A \rightarrow \Diamond B, A \wedge B, \ldots, B(y)\}$$

$$comp\ (x) = \{\mathrm{A}, A \rightarrow B, A \rightarrow \Diamond B, A \wedge B, \ldots\} \qquad comp\ (y) = \{\Box(x)\}$$

A transformation is intended to replace a pattern instantiation by the instantiation of another pattern. The resulting specification should either have the same semantics as the original specification, or it should be a refinement.

A *transformation* rule $sp \rightarrow_c tp$ consists of

- a *source pattern sp* on the left-hand side,
- a *target pattern tp* on the right-hand side,
- a relation *refine,* and
- a relation *equiv.*

The relation *refine* determines which instantiations of the source pattern may be replaced by which instantiations of the target pattern. If we define $\psi$ as the function that represents the application of a transformation, we get the following situation:

$$\begin{tikzcd}
\mathrm{S} & p.c \arrow[r, "\psi"] \arrow[d, "sem"'] & \psi(p).\psi(c) \arrow[d, "sem'"] & \mathrm{S'} \\
& M \arrow[r, phantom, "\supseteq"] & M' &
\end{tikzcd}$$

The relation *refine* must guarantee that $sem'\ (\psi(p).\psi(c)) \subseteq sem(p.c)$. The case of refining transformations is considered in an extended example in Section 3. The transformations in this example use constraints for the finite representation of the relation *refine*.

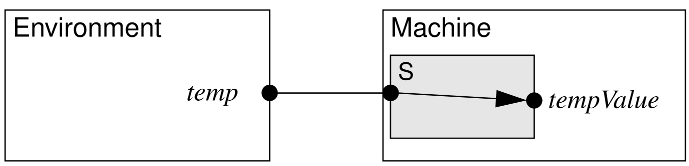
In the case of a semantic invariance, $\psi$ is constructed with the relation *equiv* and the condition $sem'\ (\psi(p).\psi(c)) = sem(p.c)$. Section 4 gives an example for a semanticly invariant transformation, that is used for restructuring of the specification.

If additionally the specification techniques for target and source patterns are identical, the diagram is simplified as follows:

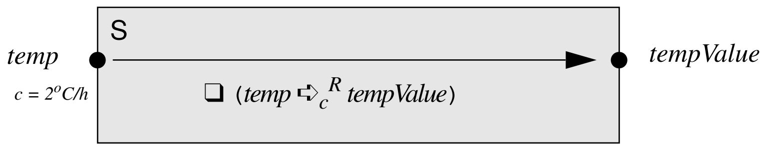
$$\begin{tikzcd}
\mathrm{S} & p.c \arrow[rr, "\psi"] \arrow[dr, "sem"'] & & \psi(p).c \arrow[dl, "sem"] & \mathrm{S} \\
& & M & &
\end{tikzcd}$$

## 3 Refinement of Specifications

The following example shall illustrate how an abstract requirement specification can be refined to an implementable specification by instantiation of patterns and by application of transformations: The requirement specification demands that a certain part S of a building automation system shall guarantee that the outdoor tempereture *temp* and its machine representation *tempValue* diverge by at most $1^{o}$ Celsius.

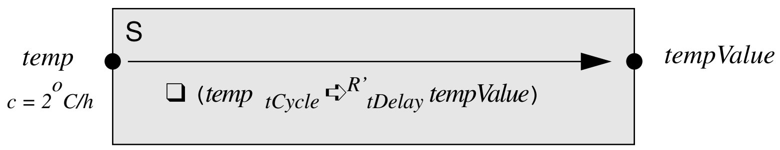



The specification of this requirement has to correlate an input quantity *p* and an output quantity *q* with respect to a relation *R*. This is performed by an instantiation of the *FOREST* specification pattern *SluggishSourceCopy (p, q, R, c)* (see Fig. 1). To this purpose, the real environmental temperature *temp* would be inserted as input *p* and the measured, transported value `tempValue` as ouput *q*. The relation *R* expresses the tolerable fault caused by measurement, quantization and transport, in our case *(temp, tempValue)* $\in$ *R* iff *temp*-$1^{o}$C < *tempValue* < *temp*+$1^{o}$C, i.e. a maximal deviation of $1^{o}$ C .




Additionally, the pattern demands that the change rate of the input *p* is bounded by *c*. Because the instantiation uses the outdoor temperature `temp` as input quantity, it is possible to limit the input change rate *c* by *2* $^{o}$*C/h* . You should note, that this information is very important for the further system refinement. The automatic introduction of relevant informations is only possible through the pattern-based specification style.

The application of the *SluggishSourceCopy* pattern supports the precise and abstract specification of the required behaviour, but it gives no advice, how to move towards a first *system design*. Apparently, a regular, sufficiently frequent and fast measurement (and transport) of the input *p* would satisfy the requirement, because the input change rate is limited. In fact, it is possible to derive a maximal time interval $\Delta t$ for each pattern instantiation (with a good-natured relation *R*). If the input quantity is measured and transported at least once within this time interval, the requirement is fulfilled. Obviously, the derived behaviour can then be specificed as a pattern, too: The *CyclicDelayedCopy (p, q, R, tCycle, tDelay)* pattern (Fig. 1) expresses, that the input quantity *p* is measured at least every *tCycle* time units and a related value $\in$ *R(p)* is assigned to *q* with a maximal time delay of *tDelay.*

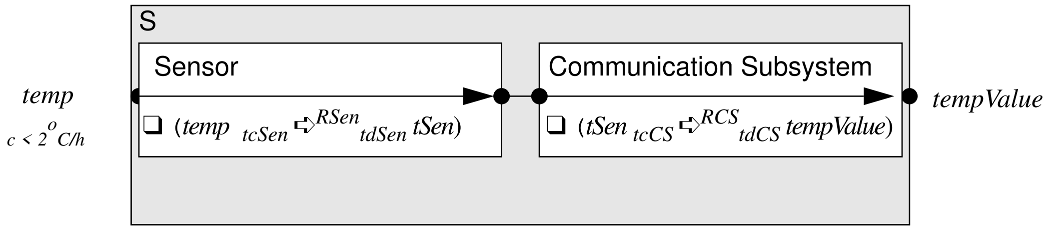



Constraints: $\forall temp \in [-40,70] : R'(temp) \subseteq Q(temp, \Delta t) \neq \emptyset$

$\Delta t = tCycle + tDelay$

So far, we have replaced one pattern-pased fragment of the specification by another. The transition from the *SluggishSourceCopy* pattern to the *CyclicDelayedCopy* pattern represents a non-trivial experience, that can be formally stored in a *transformation* for later reuse. The transformation *SluggishSourceToCyclicDelay* (see Fig. 2) specifies under which conditions an instantiation of a *SluggishSourceCopy* pattern may be replaced by an instantiation of a *CyclicDelayedCopy* pattern. The main part of this transformation determines proper constraints for the cycle and delay times and the input-to-ouput relation. The constraint a) requires that for each input *p* the new relation *R'* provides an output $q \in R'(p)$, that is corresponding (w.r.t. to R) with all subsequent inputs that are possible after the input *p* within $\Delta t$. Little values for the sum $\Delta t$ of cycle and delay times allow a less restricted, derived relation *R'* , and conversely, if the times become too large, the relation will be empty.

As a next step towards implementation we choose to split the system S into a sensor component, that is responsible for the measurment of *temp* and a commuication subsystem, that is responsible to transport the sensor output to the desired location of *tempValue*.To this purpose, the specification of S must be distributed among the new components. Again, the transformation *DistCyclicDelay* (see Fig. 3) recommends to instantiate *CyclicDelayedCopy* patterns for both components.




Constraints: $\forall temp \in [-40,70] : R'(temp) \subseteq Q(temp, \Delta t) \neq \emptyset$

$\Delta t = tCycle + tDelay$

$RSen \circ RCS \subseteq R'$

$tcSen, tcCS \leq tCycle$

$tdSen + tdCS \leq tDelay$

The sensor component could be further refined to an existing sensor with a $0.1\ ^oC$ measuring error, i.e. $RSen = \{ (temp, tSen) \mid temp-0.1 \leq tSen \leq temp+0.1 \}$, a measuring cycle of $tcSen = 0.5\ s$ and a delay of $tdSen = 1\ s$. Using all constraints and the available instantiation information, it is possible to derive certain restrictions for the development of the communication subsystem (which is intended to transport, but not to modify data, i.e. $RCS$ = id):

- To cope with $RSen \circ \mathrm{id} \subseteq R'$ we set $R' := RSen$
- It is $R = \{ (temp, tempValue) \mid temp-1 \leq tempValue \leq temp+1 \}$.
  Therefore, it is $Q(temp, \Delta t) = [\, temp+2\Delta t-1,\ temp-2\Delta t+1 \,]$
- To satisfy $Q(temp, \Delta t) \neq \emptyset$ , it must be $p+2\Delta t-1 \leq p-2\Delta t+1$, so that $\Delta t \leq 0.5\ h$.
  To satisfy $R(temp) \subseteq Q(temp, \Delta t)$ i.e. $[temp-0.1,\ temp+0.1] \subseteq [\, temp+2\Delta t-1,\ temp-2\Delta t+1 \,]$, it has to be $\Delta t \leq 0.45\ h$.
  Therefore we use the maximal $\Delta t := 0.45\ h$.
- Since $tcCS \leq tCycle = \Delta t - tDelay = \Delta t - tdSen - tdCS$ , the cycle and delay times of the communication subsystem are restricted:
  $tcCS \leq (0.45\ h - 1s) - tdCS = 1619\ \mathrm{s} - tdCS$.

Regarding the cycle and delay times as QoS-parameters, the development or selection of a proper communication system becomes a straightforward task. It is not always possible to automatically derive such parameters and constraints from the application of transformations. Nevertheless, patterns and transformations introduce only relevant quantities and statements. Thus, they help to concentrate on the essential aspects of the current development.

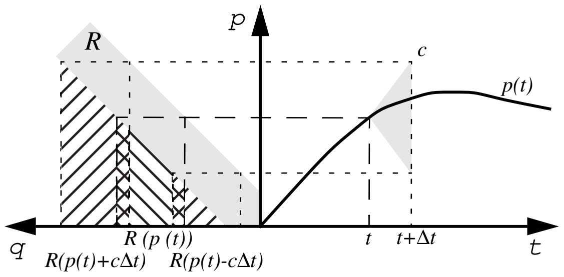
## Figure 1

**Pattern** *SluggishSourceCopy (p, q, R, c)*

*Intention* : *q* is depending on a restricted *p* with respect to *R*.

*Definition* : $\Box(p \Rightarrow_c^R q)$, where $\Box(p \Rightarrow_c^R q) =_{Df}$ $\Box(p,q) \in R \wedge \Box(|dp/dt| < c)$

*Illustration* :

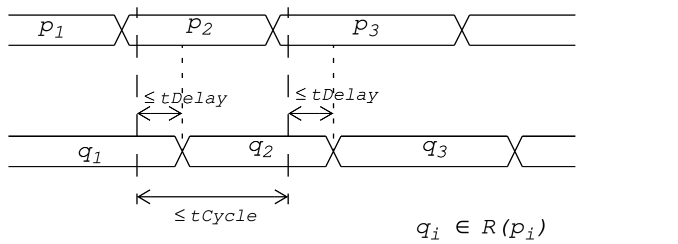



**Pattern** *CyclicDelayedCopy (p, q, R, tCycle, tDelay)*

*Intention* : *p* is regularly measured and copied to *q* as *R(p)* with respect to a transport delay.

*Definition* : $\Box(p\ {}_{tCycle}\!\Rightarrow^R_{tDelay}\ q)$ is defined as FIFO with time bounds

*Illustration* :




## Figure 2

**Transformation** *SluggishSourceToCyclicDelay*

Intention :

Definition : *SluggishSourceCopy (p, q, R, c)* $\longrightarrow$ *CyclicDelayedCopy (p, q, R', tCycle, tDelay)*

Constraints : Let $Q(p,\delta t) := \bigcap_{p' \in [p - c\cdot\delta t,\, p + c\cdot\delta t]} R(p')$

Let $\Delta t := tCycle + tDelay$

a) $\forall p \in P: R'(p) \subseteq Q(p, \Delta t) \neq \emptyset$

Illustration :

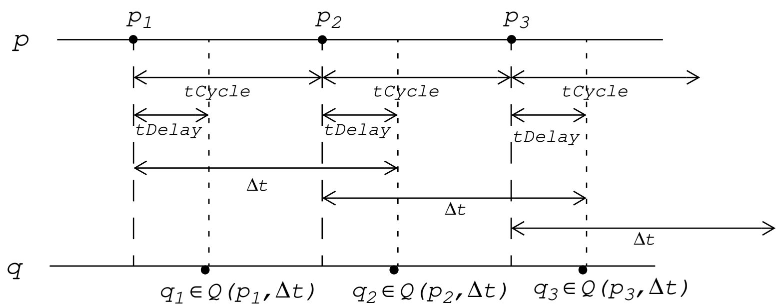
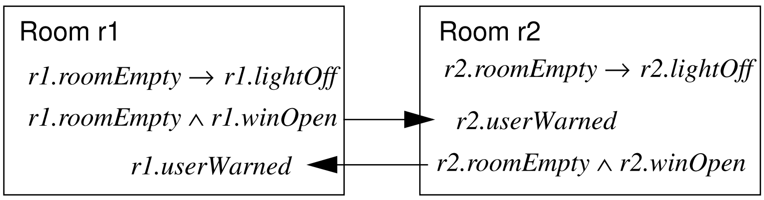


**Figure 3**

| **Transformation** *DistCyclicDelay* | | |
|---|---|---|
| Intention | : | Distributes the cyclic delay among two components |
| Definition | : | *CyclicDelayedCopy (p, q, R, tCycle, tDelay)* $\longrightarrow$ <br> *CyclicDelayedCopy (p, r,* $R_1$*,* $tCycle_1$*,* $tDelay_1$*)* $\wedge$ <br> *CyclicDelayedCopy (r, q,* $R_2$*,* $tCycle_2$*,* $tDelay_2$*)* |
| Constraints | : | a) $R_1 \circ R_2 \subseteq R$ <br> b) $tCycle_2 \leq tCycle_1 \leq tCycle$ <br> c) $tDelay_1 + tDelay_2 \leq tDelay$ |

## 4 Views on a System

The structure of a system specification often follows a certain scheme to achieve a uniform and intelligible access, e.g. the building structure in case of a building automation system. But in many cases, the selected structure is arbitrary and could be replaced by another. Because patterns express certain global properties, they have the potential to represent another structural view on the system. Different views on the same system can significantly increase the comprehensibility of the system specification. On the other hand, the restructured view may be not expressible by the original description technique. In this case, the explicit usage of patterns is the only way to support an alternative view. In this section we will consider a simple example to illustrate how the clean switching between the geographical view and the functional view on a system can be formalised by a semanticly invariant transformation.




Each of the two neighboured rooms *r1* and *r2* has a specification for normal operation and for critical operation. Since both component specifications are exclusively created as instantiations of the pattern *Room,* the contained specification types are completely known.

| **Pattern** *Room (r, r')* | | |
|---|---|---|
| *Intention* | : | Specifies the behaviour of a room with a neighbour. |
| *Definition* | : | *r.roomEmpty* $\rightarrow$ *r.lightOff* $\wedge$ <br> *r.roomEmpty* $\wedge$ *r.winOpen* $\rightarrow$ *r'.userWarned* |

| **Pattern** *Normal (r1, r2)* /* *Error (r1,r2)* is similar */ | | |
|---|---|---|
| *Intention* | : | *Normal* selects the normal operation specifications of its parameters. |
| *Definition* | : | r1.roomEmpty $\rightarrow$ r1.lightOff $\wedge$ <br> r2.roomEmpty $\rightarrow$ r2.lightOff |

| **Transformation** *RestructureRooms* | | |
|---|---|---|
| Intention | : | Restructures the specification of a room into normal and error specifications. |
| Definition | : | *Room (r1,r2)* $\wedge$ *Room (r2,r1)* $\longrightarrow$ *Normal (r1, r2)* $\wedge$ *Error (r1, r2)* |
| Constraints | : | *none* |

The application of the transformation *RestructureRooms* leads to a semanticly equivalent specification, that is restructered by a functional point of view. In this case, the transformation needs no constraints to guarantee the semantic invariance.

Normal (r1,r2)

$r1.roomEmpty \rightarrow r1.lightOff$

$r2.roomEmpty \rightarrow r2.lightOff$

Error (r1,r2)

$r1.roomEmpty \wedge r1.winOpen \rightarrow r2.userWarned$

$r2.roomEmpty \wedge \rho2.winOpen \rightarrow r1.userWarned$

## 5 Summary and Future Work

Obviously, transformations can provide valuable knowledge for the system development process - even if a specification is not completely expressible by pattern instantiations. The known pattern-based approaches concentrate on the discovery, definition and reuse of patterns to improve specification *products*. However, transformations try to support the software development *process*. Since there is a close relation between patterns and transformations, the introduction of transformations into pattern-based specification techniques seems to be a promising next step, that still raises a number of questions:

- **Support for the evaluation of constraints:**
  So far, it is not clear if an automatic or at least semi-automatic application of the proposed transformations is feasible. The purely syntactical application of several transformations leads to a number of constraints, that can be regarded as a system of inequations, for which a solution has to be found. If the domains are finite it should be possible to use fixpoint iteration techniques to compute such a solution.
- **Existence of a complete coverage:**
  It is desirable to apply as many patterns and transformations as possible to increase the degree of reuse. But is it possible (at least in a resctricted application domain) to find a set of patterns and transformations, that allow a completely pattern-based specification of a domain-typical problem, i.e., does there exist a coverage of the specification by such patterns?
- **Transformations between different FDTs:**
  There are various application scenarios for transformations. For instance, source and target patterns can be specified in different FDTs to enable the transition from a property-oriented to an operational specification. In this case, the transformation hides the details concerning the transition of the semantics. Especially the transition from the communication service specifications in temporal logic to the SDL protocol specification patterns used in subproject B4 of the SFB 501 will be the subject of further considerations.

## References


[AlHe91] R. Alur, T.A. Henzinger: *Logics and Models of Real Time: A Survey,* in J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg (eds.), Real-Time: Theory and Practice, LNCS 600, 1991

[Ga+94] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns - Elements of Reusable Object -Oriented Software*, Addison-Wesley, 1995

[GeGoRö97] B. Geppert, R. Gotzhein, and F. Rößler: *Configuring Communication Protocols Using SDL Patterns*, 8th SDL Forum, SDL'97, Paris/Evry, France, 1997

[GoKrPe98] R. Gotzhein, M. Kronenburg, C. Peper: *Reuse in Requirements Engineering: Discovery and Application of a Real-Time Requirement Pattern*, 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRFT'98), Lyngby, Denmark, 1998, pp. 65-74

[KrPe99a] M. Kronenburg, C. Peper: *An Example of a FOREST Problem Specification*, SFB 501 Report 01/1999, University of Kaiserslautern, Germany

[KrPe99b] M. Kronenburg, C. Peper: *Definition and Instantiation of a Reference Model for Problem Specifications*, accepted for SEKE'99

[KrPe99c] M. Kronenburg, C. Peper: *FOREST,* http://rn.informatik.uni-kl.de/~forest

[KrGoPe96] M. Kronenburg, R. Gotzhein, C. Peper: *A Tailored Real-Time Temporal Logic for Building Automation Systems,* SFB 501, Technical report 16/96, University of Kaiserslautern, Germany, 1996

[PeGoKr97] C. Peper, R. Gotzhein, M. Kronenburg: *A Generic Approach to the Formal Specification of Requirements*, 1st IEEE International Conference on Formal Engineering Methods 1997 (ICFEM'97), Hiroshima, Japan