Non-refinement Transformations of Software Architectures

Andreas Kerschbaumer
Institute for Software Technology
Graz University of Technology
kerschbaumer@ist.tugraz.at

January 2002

Abstract

This paper deals with the refinement of software architectures. Using an example transformation of an architectural pattern from a call-return style to a batch-sequential one, it is shown that sometimes an advancement of a software architecture is not a correct refinement relation in the formal sense. However formal analysis may and should be used to argument about the changed and unchanged properties. The patterns are modeled and analyzed using the specification language LOTOS, the temporal logic ACTL and the CADP toolbox.

1 Introduction

Classical formal software development proceeds from an abstract specification through several steps to a final implementation where each development step needs to fulfill a formal refinement relation to its preceeding step. For example VDM [7], B [1], and Z [17], which are mainly used for the specification of sequential systems, all define refinement on the notion of a retrieve relation. Examples for concurrent specification languages are process algebras like CSP [6] or LOTOS [16] which in general define refinement on the basis of observable actions.

This paper presents one result of our ongoing work in the field of refinement of software architectures. A software architecture describes the gross structure of a system: its components (and subsystems) and their connections [15]. Several architectural styles have been identified [15] and several views may be taken when specifying an architecture [9]. Architecture definition languages (ADL's) have been developed for the precise specification of software architectures, among of which are Wright [2], PADL [3] and Rapide [10]. They all define architectures in the process view using process algebras or a comparable specification formalism, concentrating on the components and their interactions and neglecting functional behavior (i.e. the input-output behavior) as far as possible. We are also taking the process view in our work, however not defining a new ADL.

Generally, architectural specifications are directly derived from the requirements specification. In a first step only the major requirements may be used for a first architectural model. This model will then have to be refined taking further requirements into consideration. This may result in a completely restructured model, where even the global structure may have to be changed and, worse, the restructured model is not a formal refinement of its predecessor.

These considerations are elaborated in the following sections using an example. A transformation from an architectural pattern based on the call-return style to a pattern in the

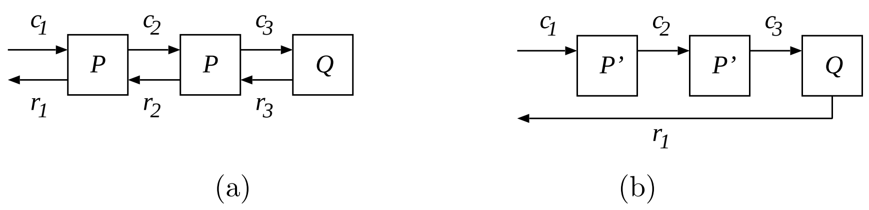

Figure 1: The structure of (a) the call-return pattern and (b) the batch-sequential pattern.

batch-sequential style is proposed. It is shown that the batch-sequential pattern behaves differently and is not a refinement, what indeed is the intention of the transformation: an increase in efficiency. Finally the results are discussed.

2 A Transformation of an architectural pattern

The architectural patterns are modeled in LOTOS [16] using the XTL tool [11] of the CADP toolbox [5] for model-checking properties which are formalized as ACTL (Action based CTL) formulas [13].

The abstract pattern (Fig. 1(a)) is a chain of three components using the call-return style offering the interface channels c_1 and r_1. Such a pattern can be frequently found in multi-tiered client-server systems:

```
hide $c_2, r_2, c_3, r_3$ in
    $P[c_1, c_2, r_1, r_2]$ |[$c_2, r_2$]| $P[c_2, c_3, r_2, r_3]$ |[$c_3, r_3$]| $Q[c_3, r_3]$
```

After being called, the components P do some calculation and then call their neighbor. They may perform different computations, but this is not relevant for the architectural model. Their interaction behavior is the same; they all have the same architectural model. Component Q finishes the chain and returns a result which is forwarded back through the chain without a modification. The components are modeled as LOTOS processes:[1]

```
process $P[c_1, c_2, r_1, r_2]$ : noexit :=
    $c_1$ ? $x : Msg$;                (* receive call *)
    choice $y : Msg$ []               (* compute *)
        i; $c_2$ ! $y$;               (* send call *)
        $r_2$ ? $z : Msg$;            (* receive return *)
        $r_1$ ! $z$;                  (* send return *)
        $P[c_1, c_2, r_1, r_2]$       (* next cycle *)
endproc

process $Q[c, r]$ : noexit :=
    $c$ ? $x : Msg$;                  (* receive call *)
    choice $y : Msg$ []               (* compute *)
        i; $r$ ! $y$;                 (* send return *)
        $Q[c, r]$                     (* next cycle *)
endproc
```

A disadvantage of this pattern is, that all processes P are locked until the computation has passed down the chain and returned a result. The architecture would be more efficient if a P is ready for the next computation right after the call down the chain. Such an improvement can be made by changing the architectural style from a call-return to a batch-sequential one (Fig. 1(b)):

```
hide $c_2, c_3$ in
    $P'[c_1, c_2]$ |[$c_2$]| $P'[c_2, c_3]$ |[$c_3$]| $Q[c_3, r_1]$
```

[1]The internal action **i** makes the choice an internal one. We use internal choice to model an arbitrary internal computation.

A component P' now only calls its neighbor in the chain, does not get a return result and is ready for the next computation:

```
process $P'[c_1, c_2]$ : noexit :=
  $c_1$ ? $x : Msg$;              (* receive call *)
  choice $y : Msg$ []             (* compute *)
    i; $c_2$ ! $y$;               (* send call *)
    $P'[c_1, c_2]$                (* next cycle *)
endproc
```

It can easily be seen that the two patterns are not related by a classical refinement relation, regardless of the underlying semantics: even in the trace semantics[2] the batch-sequential pattern is not a refinement of the call-return pattern.[3]

What happened? Essentially one property does not hold any more in the concrete batch-sequential pattern, namely that it is not possible to make a call on channel c_1 until the preceding call to c_1 has been answered at r_1. This can be formalized in ACTL and be model-checked using XTL:

$$\mathbf{AG}\ ([c_1]\ \mathbf{A}[\mathbf{true}_{\neg c_1}\ \mathbf{U}_{r_1}\ \mathbf{true}]) \tag{1}$$

Exactly this change has been intended to improve efficiency. Other properties remain valid, e.g. that each call to c_1 is eventually answered on r_1 (2), or that return messages from Q are eventually forwarded without a change (3):[4]

$$\mathbf{AG}\ ([c_1]\ \mathbf{A}[\mathbf{true}_{\mathbf{true}}\ \mathbf{U}_{r_1}\ \mathbf{true}]) \tag{2}$$

$$\forall\, x : Msg \bullet\ \mathbf{AG}\ ([r_3\ !\ x]\ \mathbf{A}[\mathbf{true}_{\neg r_1}\ \mathbf{U}_{r_1\ !\ x}\ \mathbf{true}]) \tag{3}$$

3 Discussion and related work

One could argue that the transformation above is not necessary or could be avoided when the abstract specification is the batch-sequential pattern. The call-return pattern would then be an inappropriate formalization of the architecture, not incorporating enough (or the right) requirements.

But when developing an architecture it is initially not that clear what the relevant requirements are. Furthermore, usually not all requirements are completely known when the first architectural model is derived, especially when using an iterative development process. Finally additional requirements may be added in the evolution of the system. Therefore, changes like the one presented above are not unlikely to occur in practice.

The work most closely related to ours is [12] where refinement transformations of architectural schemas, comparable to our patterns, are formally defined. However the refinements are solely defined on the structure and neglect the architectural behavior. Therefore behavioral non-refinements cannot be detected using this formalization.

The ADL's Wright [2], PADL [3] and Rapide [10] also have a notion of refinement. This refinement only permits the exchanging of a component but not the change of the overall structure. The new component has to *conform* to the abstract specification (or to the role

[2]Which is the weakest of all process semantics and cannot even distinguish between external and internal choice.

[3]But vice versa.

[4]Property (3) is given for the call-return pattern, for the batch-sequential pattern it is trivially true.

in the case of Wright). Simultaneous structural and behavioral transformations, as the one presented above, are not allowed.

Jones proposes an efficiency transformation similar to our example for his object based design notation $\pi o\beta\lambda$ [8]. He formalized $\pi o\beta\lambda$ in the π-calculus and tried to prove formally that the transformation is a correct refinement. The proof failed. We think that it *has* to fail because the patterns are *not* behavioral equivalent.

Finally, there has lately been some work in the field of architectural patterns [4, 14], however without a precise specification and without analyzing possible transformations.

4 Conclusion

An architectural model usually has some implementation bias; it does not only reflect the intended requirements but has also additional architectural properties inherent to its structure and its behavior. Some of these properties may not be of importance for the system in question or even not be desired.

When putting further requirements into consideration the architecture may have to be changed completely, such that it does not establish a formal refinement relation to its predecessor; some of the architectural properties may have changed. Using predefined transformations with known (and proved) preservations and non-preservations of properties could help in making such restructuring decisions. Such architectural development steps, where the structure of the system changes, may then not be related by refinement any more. Of course, in the development of the components of an architecture classical refinement is still of great value.

Acknowledgments

I would like to thank Peter Lucas and Bernhard Aichernig for the valuable discussions. Special thanks to INRIA, CNRS and Verimag for providing a free research license of their CADP tools.

References

[1] J.-R. Abrial. *The B-Book.* Cambridge University Press, 1996.

[2] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–49, 1997.

[3] M. Bernardo, P. Ciancarini, and L. Donatiello. On the formalization of architectural types with process algebras. In D. S. Rosenblum, editor, *Proc. of the 8th ACM Int. Symp. on the Foundations of Software Engineering (FSE-8)*, pages 140–148. ACM Press, November 2000.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented Software Architecture: A System of Patterns.* John Wiley & Sons, 1996.

[5] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A protocol validation and verification toolbox. In *Proceedings of the 8th*

Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA), pages 437–440, August 1996.

[6] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[7] C. B. Jones. *Systematic Software Development using VDM.* Prentice-Hall, second edition, 1990. Out of print.

[8] C. B. Jones. Process-algebraic foundations for an object-based design notation. Technical Report UMCS-93-10-1, Department of Computer Science, University of Manchester, 1993.

[9] P. Kruchten. The "4+1" view model of software architecture. *IEEE Software*, 12(6):42–50, November 1995.

[10] D. C. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, 1995.

[11] R. Mateescu and H. Garavel. XTL: A meta-language and tool for temporal logic model-checking. In *Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, July 1998.

[12] M. Moriconi, X. Qian, and R. A. Riemenschneider. Correct architecture refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, April 1995.

[13] R. D. Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science (La Roche Posay, France)*, volume 469 of *Lecture Notes in Computer Science (LNCS)*, pages 407–419. Springer-Verlag, 1990.

[14] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-oriented Software Architecture: Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons, 2000.

[15] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall, 1996.

[16] K. J. Turner, editor. *Using Formal Description Techniques.* John Wiley & Sons, 1993.

[17] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall, 1998.