Proceedings of the Fifth Nordic Conference on Pattern Languages of Programs

Edited by Aino Vonge Corry, Pavel Hruby and Kristian Elof Sørensen

Published by REA Technology
ISBN 978-87-92411-00-6

Viking PLoP 2006, Proceedings of the Fifth Nordic Conference on Pattern Languages of Programs,
edited by Aino Vonge Corry, Pavel Hruby and Kristian Elof Sørensen.

For more information about Viking PLoP please visit http://vikingplop.org or http://vikingplop.net.
For more information about Hillside Group and patterns in general please visit http://hillside.net.

Published by REA Technology
ISBN 978-87-92411-00-6

Table of Contents

Introduction

Patterns and pattern languages are ways to describe best practices, good designs, and capture experience in such a way that it is possible for others to reuse it.

Building upon the traditions and values of the patterns community, a Nordic Conference on Pattern Languages of Programs is held in Scandinavia every year to enable people in Scandinavia, who might otherwise not attend another PLoP conference, to learn about patterns. The First Nordic Conference on Pattern Languages of Programs, Viking PLoP 2002 was held in Højstrupgård, Denmark, followed by Viking PLoP 2003 in Bergen, Norway, Viking PLoP 2004 in Uppsala, Sweden, and Viking PLoP 2005 in Helsinki, Finland.

Viking PLoP 2006, the Fifth Nordic Conference on Pattern Languages of Programs was held from September 28 to October 1 at Højstrupgård Castle near Helsingør, 60km north of Copenhagen, Denmark.

At writers' workshops 9 papers were discussed, from which 8 have been selected for these proceedings: 5 software patterns, 2 patterns in other fields and 2 papers on applying patterns.

We would like to express our thanks to participants of the writers' workshops for providing useful feedback to the authors and thus helping to improve their papers for these proceedings; Rebecca Rikner for managing the Nordic Design Patterns Association, the legal entity behind the conference; Jim Coplien for being a game master at Viking PLoP 2006, and the personnel at Højstrupgård Castle for fulfilling our culinary wishes.

In February 2007,
Aino Vonge Corry, Pavel Hruby and Kristian Elof Sørensen

Conference Organization

Program Chair
Aino Vonge Corry

Conference Chairmen
Kristian Elof Sørensen and Pavel Hruby

Program Committee
Juha Pärssinen, Uwe Zdun, Kevlin Henney, James O. Coplien and Klaus Marquardt

Shepherds
Jorje Ortega Arjona , James O. Coplien, Richard Gabriel , Neil Harrison, Cecilia Haskins, Michael Kircher, Andreas Rüping, Peter Sommerlad, Michael Weiss, and Uwe Zdun.

Writers' Workshops Participants
Aino Vonge Corry, Andreas Rüping, Birgit Zimmermann, Cecilia Haskins, Eric Evans, James Siddle, Jayashree Kar, Jim Coplien, Juergen Salecker, Juha Pärssinen, Klaus Marquardt, Kristian Elof Sørensen, Lars-Helge Netland, Met-Mari Nielsen, Michael Weiss, Pavel Hruby, Rebecca Rikner, Susanne Hørby Christensen, Yngve Espelid

Shepherding Award

Shepherding award is given to the reviewer, or shepherd, who helped most to improve the quality of a patterns paper, before the paper was presented at the conference.

The trophy at Viking PLoP 2006 was a book Short Grooks, Viking Vistas, by Piet Hein, Danish scientist and poet with wide ranging interests. He is known for building a bridge between the hard technical and natural sciences, and the soft social subjects, and contributed to making Danish design become an international concept.

Andreas Rüping was the winner of the shepherding award at Viking PLoP 2006, for being the most exceptionally observant, helpful and insightful reviewer, and one whose mix of constructive criticism and encouragement was a great source of inspiration for the authors.

Applying Patterns

Formalizing Architectural Patterns with the Goal-oriented Requirement Language

Gunter Mussbacher[1], Michael Weiss[2] and Daniel Amyot[1]

[1] SITE, University of Ottawa, Ottawa, Canada

[2] SCS, Carleton University, Ottawa, Canada

Abstract. Many pattern descriptions put their emphasis on the solution to a problem rather than on often conflicting forces and how patterns balance such forces. This work uses the Goal-oriented Requirement Language (GRL) to formalize the forces of architectural patterns in a way that enables rigorous trade-off analysis while allowing the pattern user to determine the applicability of a pattern to the problem in a given context. The formalization of forces does not replace other pattern descriptions but rather complements them and relies on them to provide descriptions of the problem and solution. This work presents a description of the forces applicable in the context of architectural design, introduces how to represent patterns and forces with GRL, and then formalizes a subset of a recently published architectural pattern language.

INTRODUCTION

Patterns enable an efficient transfer of experience by documenting common solutions to recurring problems in a specific context. Much of the work on pattern documentation such as the Pattern Almanac (Rising, 2000) and pattern formalization (Taibi and Ngo, 2001) focuses on the solution domain. However, they seldom formalize the problem domain and relevant trade-offs between the various (and often conflicting) forces involved. Still, patterns need to be described and formalized in ways that enable the reader to de-

termine whether the particular solution presented is useful and applicable to his or her problem in a given context

To address this issue, we use the Goal-oriented Requirement Language (GRL) (URN Focus Group, 2003a) to formalize pattern forces and problem domains in a way that supports a rigorous trade-off analysis. We apply this GRL-based formalization to a recently published pattern language (Avgeriou and Zdun, 2005). The pattern language covers 23 architectural patterns categorized into eight groups according to their applicability to a specific architectural view.

In the following sections, we first present the background for patterns and the formalization of patterns. Then, we introduce the GRL as part of the User Requirements Notation and our explicit model of pattern forces which provides the basis for the trade-off analysis. This is followed by the formalization of a subset of the architectural pattern language to which we have applied our approach, including a brief description of the forces applicable in the domain of architectural design. Future trends and remarks conclude this paper.

FORMALIZING PATTERNS

Patterns describe a recurring problem that occurs in a specific context and its solution (Alexander, 1979). One of their most significant contributions is that they intend to make explicit the trade-offs between the forces involved. Each pattern describes the situation when the pattern can be applied in its context. The context can be thought of as a precondition for the pattern. This precondition is further refined in the problem description with its elaboration of the forces that push and pull the system to which the pattern is applied in different directions. Here, the problem is a precise statement of the design issue to be solved. Forces are design trade-offs affected by the pattern. They can be documented in various forms. One popular approach is to document the trade-offs as sentences like “on one hand …, but on the other hand …”.

The solution describes a way of resolving the forces. Some forces may not be resolved by a single pattern. In this case, a pattern often includes references to other patterns, which help resolve forces that were unresolved by the current pattern. Together, patterns connected in this way are often referred to as a pattern language. Links between

patterns can be of different types, including uses, refines, and conflicts. Patterns that need another pattern link to that pattern with *uses*. Patterns specializing the context or problem of another pattern *refine* that pattern. Patterns that offer alternative solutions *conflict*.

Current pattern representations are textual. They include the Gang-of-Four (GoF) form, the Coplien form, and the Alexandrian form. The GoF form (Gamma et al., 1994) includes sections for intent, motivation, structure, participants, and collaborations. The emphasis of this format is on the structure of the solution. However, the discussion of the forces is spread out over multiple sections, which makes it challenging for a developer to get an overview of when to apply a particular pattern and the consequences of using it.

Motivated by this drawback of the GoF form, the Coplien form (Coplien, 1996) defines a more rigid pattern structure. It includes explicit sections for forces and consequences, in which the forces and the implications of using the patterns are presented in bullet form. This provides quick access to the reasons for applying a pattern. Variations of this format have been proposed that present the forces/consequences as tables.

Recently, many pattern authors have returned to the Alexandrian pattern form (Alexander, 1979). It resolves the trade-off between the needs to have structure on the one hand, and the desire to create more easily readable pieces of literature, on the other. In practical use for documenting software designs, the Alexandrian form has been adapted to include the concept of explicit lists of forces and consequences from the Coplien form.

Nonetheless, with these formats there are still open issues: how to recognize under what conditions a given pattern should be selected, how to compare between different patterns that address the same problem, and how to integrate the consequences of applying a pattern or a combination of patterns into an integrated model. It is with these issues in mind that we propose a formalization of patterns using the Goal-oriented Requirements Language in a way that supports a rigorous trade-off analysis during the application of design patterns while maintaining the generality of the solution description.

Previously, (Araujo and Weiss, 2002) have proposed an explicit representation of the forces involved in a pattern and their inter-relationships. This representation suggests interpreting forces as functional and – mainly as – non-functional requirements, and uses the Non-Functional Requirements (NFR) framework (Chung et al., 2000) to analyze forces and the trade-offs made by a pattern.

In related earlier work, (Ong, Weiss, and Araujo 2003) derived the forces affected by a pattern through a close reading of the textual pattern description. The extended pattern representation enabled them to discover contributions made by the patterns to overall system concerns that were only implicit in their textual descriptions. Their main finding was that the contributions of each pattern to overall system concerns became much more apparent than what could be gleaned from reading the pattern descriptions alone.

Our approach is similar, at the level of individual patterns, to the work by (Gross and Yu, 2001), as well as (Chung et al., 2002). Both present ways of reasoning about patterns using NFRs. However, there are important differences. We are also concerned with the connections between patterns at the pattern language level, as well as with establishing models of the forces and their trade-offs that exist in a particular domain. As a result, we feel that our results are farther-reaching, and will lead to a more objective approach.

USER REQUIREMENTS NOTATION

The *User Requirements Notation* (URN) is a standardization effort of the International Telecommunications Union (ITU) and published in the Z.150 series of Recommendations (ITU-T, 2003). URN is a high-level modeling notation that makes use of goals and scenarios to model and analyze user requirements in a more formal way. The notation is generally suitable for the description of most types of reactive, concurrent, and distributed systems and services; e.g. telecommunications systems, e-commerce systems, agent systems, operating systems, and health information systems. URN is also a convenient notation for business process modeling and evolution (Weiss and Amyot, 2005). An overview of URN with a tutorial example from the wireless communication domain is presented in (Amyot, 2003).

URN consists of two complementary notations. *Use Case Map*s (UCMs) (URN Focus Group, 2003b) allow the specification of behavior and structure. The *Goal-oriented Requirement Language* (GRL) (URN Focus Group, 2003a) represents goals and stakeholders. Both notations are useful in the context of formalizing pattern forces and solutions, and for trade-off analysis. UCMs are particularly useful for modeling structural and behavioral aspects of the solution part of patterns (e.g. the consequences of applying patterns in terms of the architectural design of a system), whereas GRL is a valuable tool to

model pattern forces and their interactions. This work focuses on the later. GRL complements the NFR (Non-Functional Requirements) framework published in (Chung et al., 2000) with agent modeling concepts from the *i** framework (Yu, 1997). GRL captures business or system goals, stakeholder concerns, alternative means of achieving goals, and the rationale for goals and alternatives. Alternatives are evaluated in terms of their impact on goals and stakeholder concerns. The notation is applicable to functional requirements, but it is especially good for capturing and reasoning about the interaction of non-functional requirements. See Figure 1 for a summary of the main notation elements of GRL.

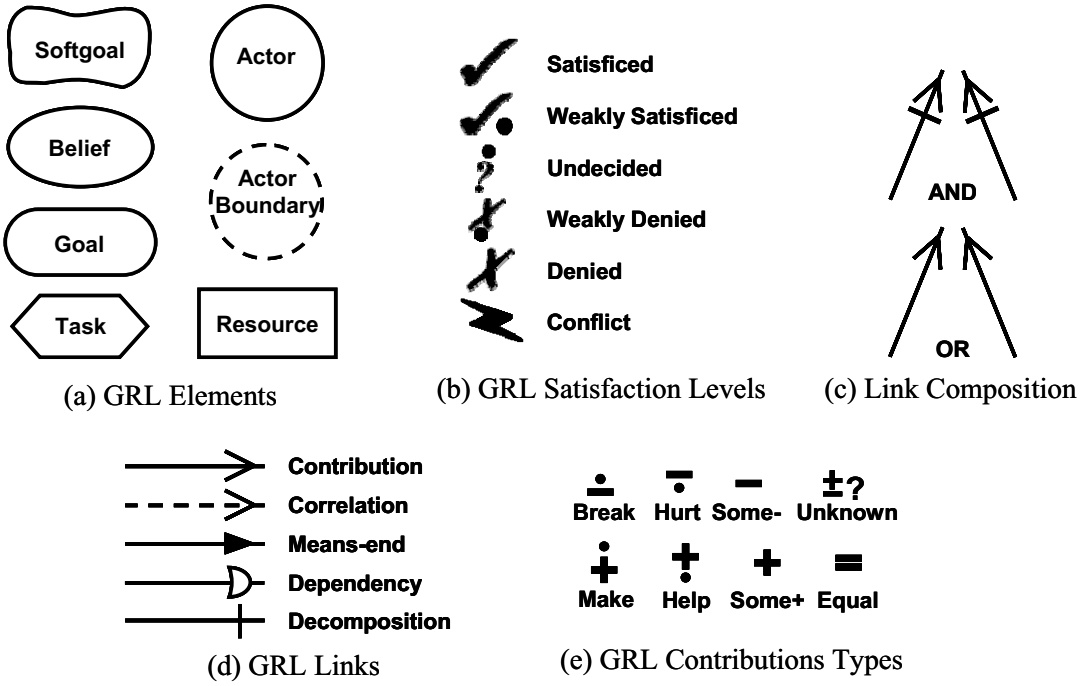


Figure 1. Summary of the GRL notation

FORMALIZING FORCES FOR TRADE-OFF ANALYSIS

We use GRL graphs to explicitly model the forces addressed by a pattern, the rationale behind them, the relationships between patterns, the contribution of a pattern to the system, and the stakeholders' interest in various aspects of the system. The GRL graph in Figure 2 shows the contributions of an individual pattern to its forces.

Forces are represented as softgoals (clouds), indicating that these cannot be achieved in an absolute manner. Patterns are modeled as tasks (hexagons), representing ways of

achieving a softgoal. The nodes of this goal graph are connected by different types of links. Direct contributions of a pattern to softgoals are shown as solid lines. Side effects (indirect contributions called correlations) are shown as dotted lines. The kind of contribution is determined by labels, indicating various degrees of positive (+) or negative (-) contributions (see Figure 1e for the complete set of labels). The complexity of the system dictates the number of levels of forces (e.g. two in Figure 2), but there is at least one level of forces.

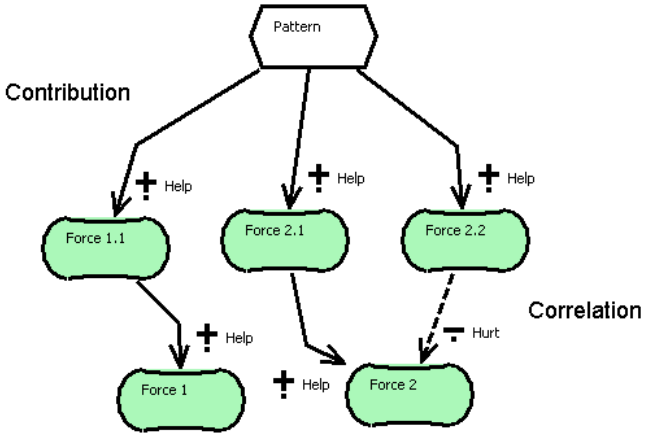


Figure 2. Modeling individual pattern – Alternative 1

Observation 1. There is only a subtle difference between forces and non-functional requirements. This difference manifests itself in non-functional requirements being at the top of the hierarchical structure of forces (i.e. at the bottom of Figure 3). The reason for this arrangement is that forces push or pull the system towards or away from non-functional requirements (e.g. better performance or greater security). As forces connect patterns and non-functional requirements, they provide an explanation of why a pattern impacts a non-functional requirement the way it does.

Therefore, model non-functional requirements also with softgoals. Show them on the same GRL graph in order to provide a visual connection between a pattern and the non-functional requirements addressed by the pattern. There is always one level of non-functional requirements in the GRL graph of an individual pattern. With tool support, the relationships between forces and non-functional requirements could be defined on a separate GRL graph and then automatically added to the GRL graphs for individual patterns in Figure 2.

Observation 2. A functional requirement can be addressed by many different alternative patterns. Each one of these patterns is described on its own GRL graph. Furthermore, functional requirements should be included in the goal hierarchy defined by patterns, forces, and non-functional requirements in order to assess impacts of patterns on functional as well as non-functional requirements.

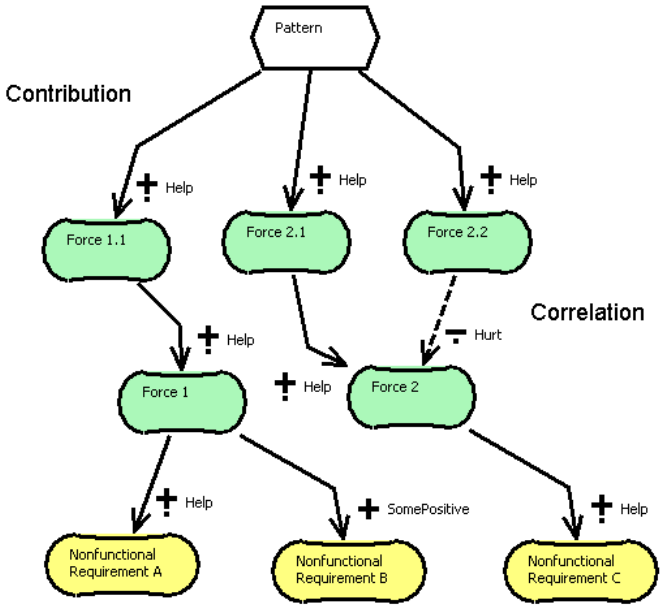


Figure 3. Modeling individual pattern – Alternative 2

Therefore, show the functional requirement on the GRL graph for individual patterns and always model the relationship between a pattern and the functional requirement to which it contributes as an OR decomposition link (barred lines). This is required due to the following reasons. On each GRL graph, the functional requirement references the same model element in GRL. Therefore, an OR decomposition has to be used instead of an AND decomposition because any one of the individual patterns can satisfy the functional requirement. An AND decomposition (even if it is distributed over many GRL graphs), on the other hand, would mean that all patterns are required in the system design before the functional requirement is satisfied. This is often the case as only a pattern combination may achieve a particular functional requirement, but it is modeled in a different way in our approach as explained later in this section.

Functional requirements are represented as (hard) goals (rounded rectangles) in order to indicate that the functional requirement can be satisfied in an absolute manner. Goals

allow us to reason about the functional requirements to which a pattern contributes. Functional requirements are shown at the top of Figure 4.

Even though our examples only show one functional requirement on the GRL graphs for individual patterns, our pattern representation is not at all limited to just one such goal. Many different goals describing many different functional requirements may be shown on a GRL graph and may be connected with decomposition links to individual patterns.

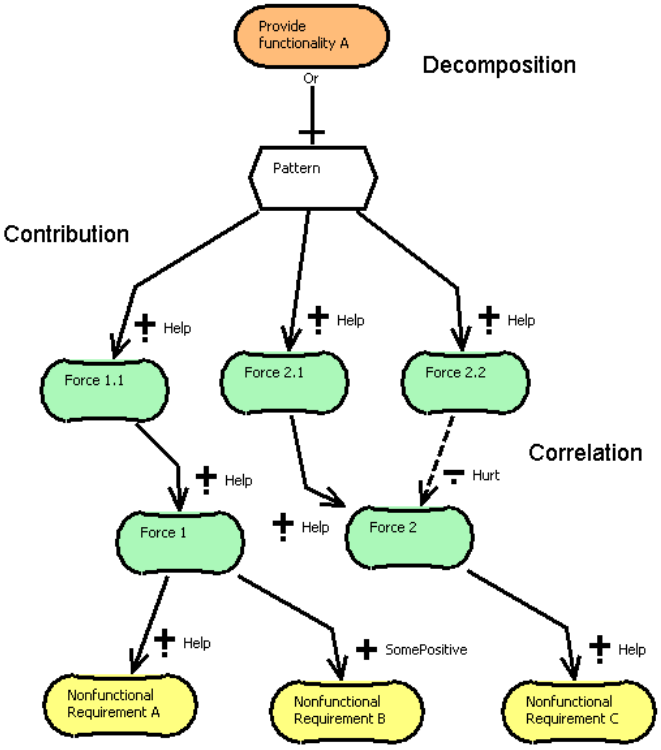


Figure 4. Modeling individual pattern – Alternative 3

Observation 3. Often, an individual pattern makes use of other patterns. The contributions of these patterns to functional and non-functional requirements have already been defined on their individual GRL graphs. They should only be created once.

Therefore, show the composition of pattern with decomposition links on the GRL graph of the individual pattern. Figure 5 constitutes the complete structure of the GRL graph for an *individual pattern*.

Decomposition links between tasks allow various relationships between patterns to be modeled (e.g. uses, refines, and conflict relationships). Figure 5 shows that “Pattern” uses “Pattern A” and “Pattern B”. The impact of “Pattern A” and “Pattern B” on forces, non-

functional requirements, and functional requirements is shown on their own individual GRL graphs. Therefore, GRL graphs for an individual pattern establish reusable models of forces applicable to many domains.

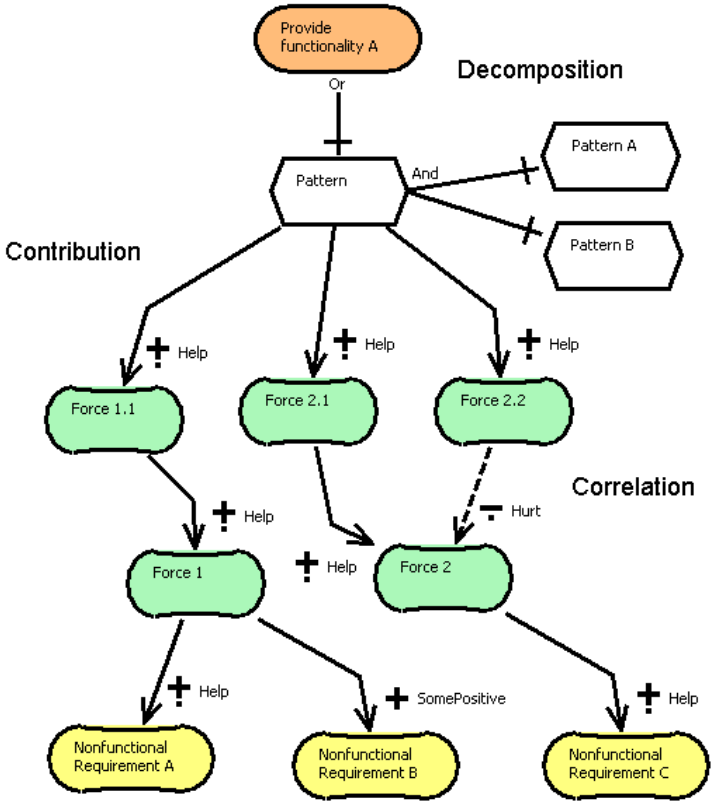


Figure 5. Modeling individual pattern

Observation 4. If a pattern combination is a pattern in itself, then the GRL graph for individual patterns is used to describe the pattern combination (see Figure 5). A pattern combination, however, may not be a pattern in itself, but may just represent the usage of several patterns together. Considering the composite pattern, there should not be a difference in our representation between an actual pattern and a pattern combination. Patterns and pattern combinations should be used interchangeably, allowing both to be considered as alternatives for a functional requirement. However, if a pattern combination is not a pattern, then there is no need to model forces and the impact of forces on non-functional requirements.

Therefore, simplify the GRL graph for an individual pattern into the GRL graph for a *pattern combination* (see Figure 6). Figure 6 addresses the point in observation 2 regarding AND decompositions of a functional requirement into many patterns ("Pattern A"

and “Pattern B”) by explicitly modeling the pattern combination as a task (“Pattern Combination”).

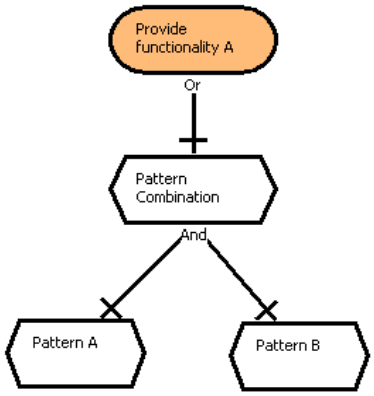


Figure 6. Modeling pattern combinations

Observation 5. A GRL graph for an individual pattern shows all forces and non-functional requirements impacted by the pattern, irrespective of the current system to be designed. Not all non-functional requirements, however, may be of interest to stakeholders in the context of any specific system design task. As it should be possible to reuse these graphs as is for each system to be designed, the stakeholders’ interest in particular non-functional requirements has to be modeled with additional GRL graphs.

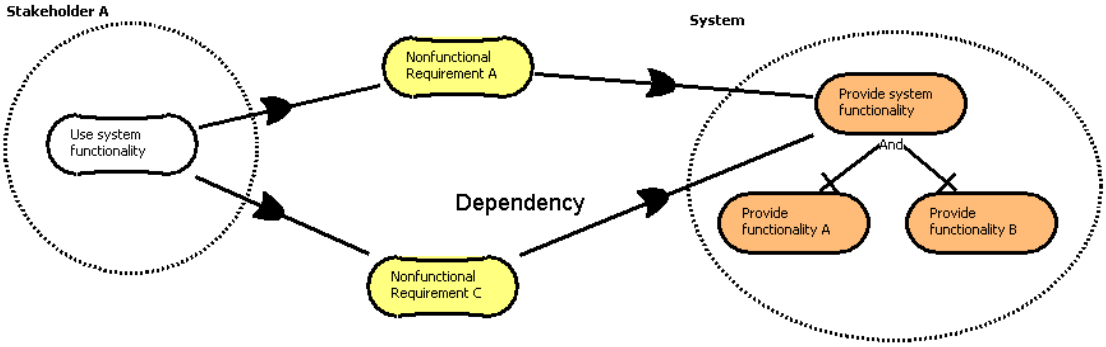


Figure 7. Modeling stakeholder concerns

Therefore, the stakeholders of a system and the system itself are modeled as GRL actors (dotted circles) with dependency links (lines with filled direction symbol) between them identifying only the non-functional requirements of interest to a particular stakeholder. The GRL graph for *stakeholder concerns* in Figure 7 shows that stakeholder A depends on the system to provide “Nonfunctional Requirement A” and “Nonfunctional

Requirement C". Note that dependency links may connect non-functional requirements with subgoals instead of the main functional goal as shown in Figure 7.

Like most goal-oriented languages, GRL supports propagation algorithms to evaluate, for a given *strategy*, to what degree the goals and softgoals in a model are satisfied (URN Focus Group, 2003a). A strategy assigns initial satisfaction values to some of the elements in the model which are then propagated to the other elements connected by contribution, correlation, and decomposition links. This enables one to make a qualitative, rapid, and global assessment of the impact of a particular choice and hence to find the most appropriate trade-off in a given context.

Observation 6. Our pattern representation consists of two hierarchies – one for non-functional requirements and one for functional requirements. The top level of both hierarchies is shown on the GRL graph for stakeholder concerns. The main functional requirements are refined into subgoals until a subgoal can be addressed by a pattern or pattern combination (e.g. one subgoal could be that the system needs to process streams of data which is addressed by patterns in the Data Flow View group of the architectural pattern language). This subgoal is then referenced on GRL graphs of applicable individual patterns or pattern combinations, and decomposed with an OR decomposition into the pattern or pattern combination, respectively. Non-functional requirements are also referenced on the GRL graphs of applicable individual patterns or pattern combinations. They are connected via contribution or correlation links to forces which in turn are eventually connected to the pattern or pattern combination, respectively.

Therefore, patterns and pattern combinations are at the bottom of both hierarchies and represent possible solutions to functional and non-functional requirements. Furthermore, the patterns and pattern combinations are also the model elements to which initial satisfaction values for the propagation algorithm should be assigned in order to evaluate alternate solutions. The initial values are propagated upwards the hierarchies to the GRL graph for stakeholder concerns where the final evaluation is shown. The two hierarchies ensure that the chosen solution is evaluated in terms of its impact on functional and non-functional requirements, avoiding cases where a solution satisfies only functional requirements at the cost of non-functional requirements or vice versa. Note that the propagation algorithm combines the contributions of several patterns or pattern combinations to a particular non-functional requirement even if the contributions to the non-functional

requirement are shown on several GRL graphs. The combined result is global and shown on each GRL graph.

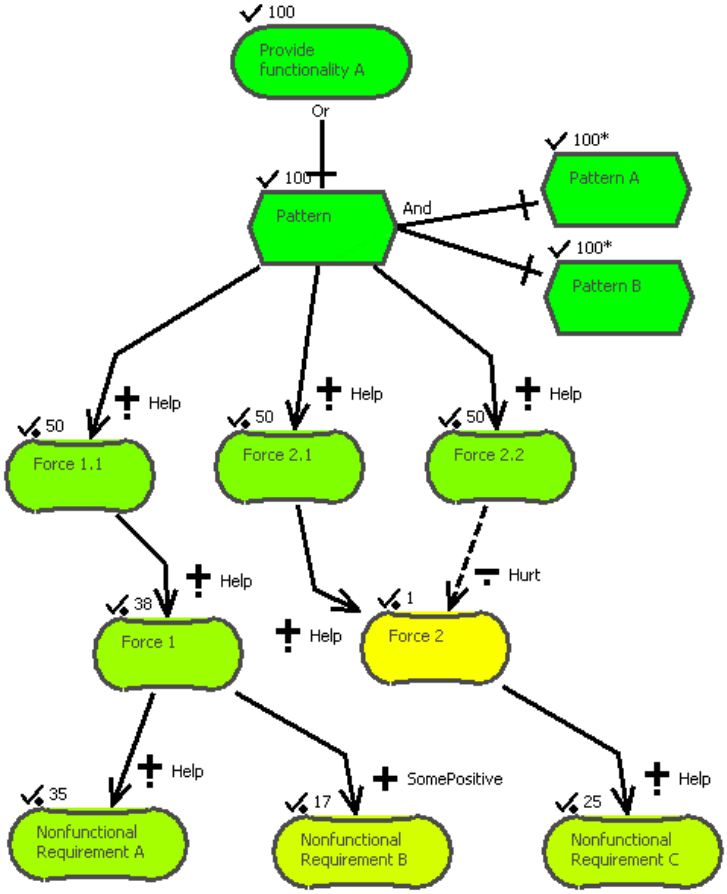


Figure 8. Investigating the impact of a pattern

Figure 8 shows the evaluation of an individual pattern with regards to its forces, functional requirements, and non-functional requirements. The evaluation results are indicated for each node of the goal graph with various checkmarks (positive) and crosses (negative) (see Figure 1b for the complete set of evaluation results) as well as numbers between -100 (denied) and 100 (satisfied). Initial values are marked with a star (*) on the evaluation diagram. Note that in order to assess the full impact of the individual pattern, all patterns used by the individual pattern (i.e., “Pattern A” and “Pattern B”) must have their initial satisfaction values set to 100. This ensures that the contributions of the patterns used by the individual pattern are taken into account for the evaluation of forces, non-functional requirements, and functional requirements. If the initial satisfaction values of “Pattern A” or “Pattern B” are changed in Figure 8, the results for those forces, non-

functional requirements, and functional requirements that also occur on the individual GRL graphs for "Pattern A" or "Pattern B" will be different.

Finally, Figure 9 shows the evaluation results from the stakeholder's point of view. The evaluation value of the main functional goal of the system is calculated (in this case it is not satisfied because only one of the two subgoals is satisfied) and the evaluation values of the non-functional requirements the stakeholder is interested in are shown. Note that these values match the values on the GRL graphs for individual patterns or pattern combinations.

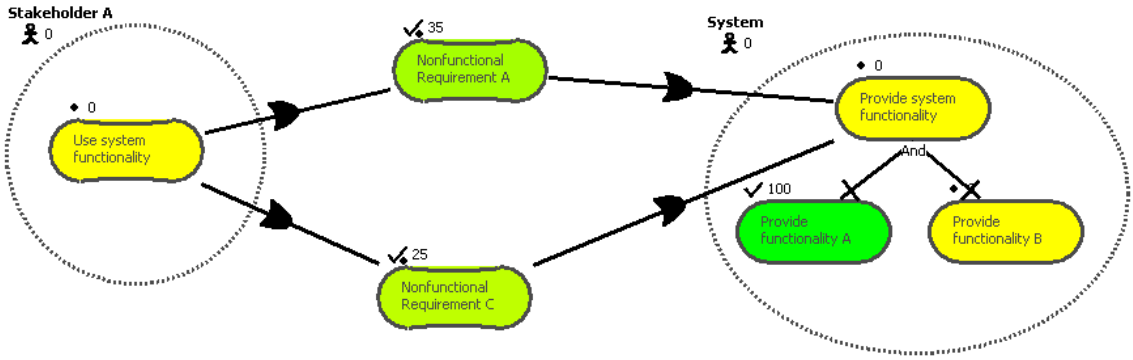


Figure 9. Assessing evaluations from the stakeholder's point of view

Observation 7. As with all modeling techniques, the usefulness of our technique presented in this section depends on how easily the technique can be included into general software development processes. A pre-condition to our technique is that regular pattern descriptions are extended to include our formalized, reusable description of pattern forces as suggested in this section (see Figure 5). Note that the presence of our GRL models does neither preclude the existence of other representations and formalizations nor does it require a particular pattern template.

Therefore, indicate what changes are most likely required to incorporate our technique into a general software development process.

1. Compile a list of stakeholders for the system. This process step should already be in place.
2. For each stakeholder, create a GRL graph for stakeholder concerns by identifying the non-functional requirements of interest to the stakeholder (see Figure 7; space permitting, more than one stakeholder may be shown on one GRL graph). The

identification of stakeholder concerns should already be a part of the process but visual modeling is an addition.

3. Refine the system's main functional goal into subgoals. This step also should be in place and again is done additionally in a visual way. Two cases have to be considered at this step:
   a. If the subgoal matches a functional goal identified in the pattern catalogue on one of the GRL graphs for individual patterns or pattern combinations, it is very likely that the subgoal does not need to be refined further. The list of goals identified in the pattern catalogue can actually be used to guide the refinement of goals.
   b. If the subgoal does not match a functional goal identified in the pattern catalogue, then again there are two choices:
      i. Further refine the subgoal until the refinement produces a matching subgoal.
      ii. Stop refinement of the subgoal and create a new pattern combination (see Figure 6) that addresses the specific need of this subgoal. A recurring, new pattern combination may even lead to a new pattern to be added to the pattern catalogue.
      iii. Stop refinement of the subgoal and address the subgoal with a solution that does not make use of patterns. The impact of this particular solution on functional and non-functional requirements can still be modeled. This, however, is only necessary if we are concerned with modeling the complete system regardless of whether patterns are used or not. If our goal is to "only" compare alternative patterns, indicate the trade-offs, and provide assistance in choosing the best pattern for the given subgoal, then we may not need to be concerned about non-pattern choices.
4. Connect the main goal or relevant subgoals with the stakeholder's non-functional requirements. In a general software development process, this is most likely not done explicitly to the extent required by our technique.
5. Determine the most suitable pattern or pattern combination by using tool-supported trade-off analysis based on GRL strategies. This is currently done in an ad-hoc manner in the worst case and with tool support in the best case.

The focus of the suggested process is on comparing patterns. The comparison simply shows the different impact the patterns have. It captures the information that is currently provided in the forces and consequences sections of a pattern description. The visualization, however, allows us to more readily see what impact the selection of a pattern or a group of patterns will have. The context for the pattern is defined by prioritizing the non-functional requirements (see step 2), thereby adapting the pattern to the environment. As indicated earlier on, our approach does not model the solution part of a pattern which is arguably the part where most of the variations happen when a pattern is used in a new context. The approach focuses on the more stable part of the forces and relationships between forces. By understanding the interaction of forces, the user is given the groundwork required to adapt the pattern in a new situation.

FORMALIZING ARCHITECTURAL PATTERNS

Observation 8. The bottom half of Figure 3 suggests that there are relationships between forces and non-functional requirements that are independent of patterns.

Therefore in this section, we formalize the relationship between forces and non-functional requirements in the domain of architectural design based on the discussion of eight non-functional requirements in chapter 2 of (Dyson and Longshaw, 2004):

1. Availability (see Figure 10) – the "working hours" of the system (the amount of time the system is up and running).
2. Performance (see Figure 11) – ability of the system to provide a timely response.
3. Scalability (see Figure 12) – ability to ensure performance when the number of users grows.
4. Security (see Figure 13) – ability to ensure access control and privacy of information.
5. Manageability (see Figure 14) – ability to monitor and adjust the system's runtime behavior.
6. Maintainability (see Figure 15) – the ease with which problems are fixed.
7. Flexibility (see Figure 16) – the ease with which new functionality and new non-functional characteristics are provided for a system.

8. Portability (see Figure 16) – the ease with which a system is migrated to a new operating environment.

Figure 10 to Figure 16 depict the forces and how each force impacts another force or non-functional requirement as mentioned in chapter 2 of (Dyson and Longshaw, 2004). The set of forces is not necessarily complete as it only reflects the forces mentioned by (Dyson and Longshaw, 2004). Other authors may want to add additional forces or relationships. The GRL graphs, however, make these forces and relationships more explicit, allowing easier identification and addition of other forces.

Reflecting the nature of forces at play for architectural design, the forces in the GRL graphs are at different levels of abstraction and cover very different things (e.g. "More robust against failure" and "Average number of users increases with time"). The GRL graphs do not prescribe a certain level of abstraction and can certainly cover very different things, because the goal of GRL graphs is to show all issues that have to be considered in a certain situation and how these issues interact with each other. The degree of contribution is used to show the importance of a force.

If the meaning of one force is not clear from its label, it is often helpful to look at the forces that contribute to the force. Alternatively, a more elaborate description or definition can be attached to the force.

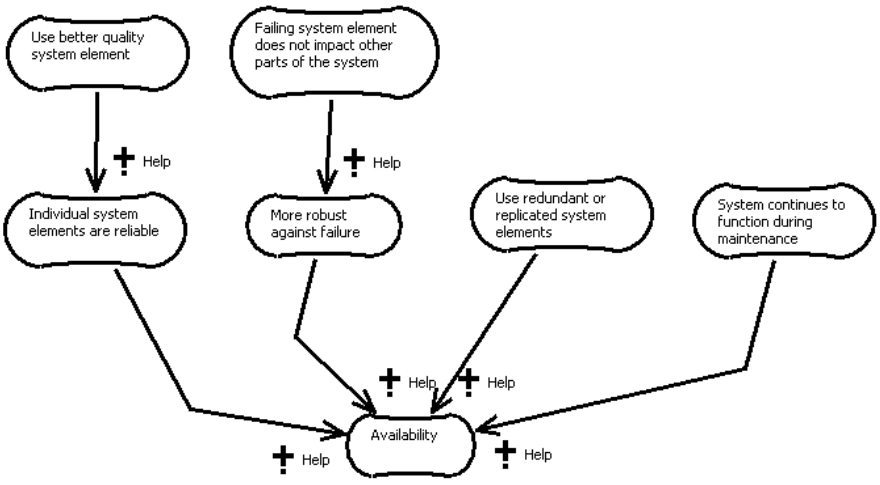


Figure 10. Forces of architectural design that impact availability

Reading the GRL graphs reveals the forces in the architectural domain. For example, Figure 10 indicates that "Use redundant or replicated system elements HELPs availability." The whole sentence expresses an invariant, consisting of a force and, in this case, its

impact on a non-functional requirement. The invariant has to be considered and understood for the domain in question. The wording of a force can be a matter of discussion. Some styles prefer the force to be worded with a keyword (similar to requirements, e.g. shall). Such styles can certainly be incorporated into our formalization. The main point is that the GRL graphs clearly identify forces, enabling a discussion on wording, etc.

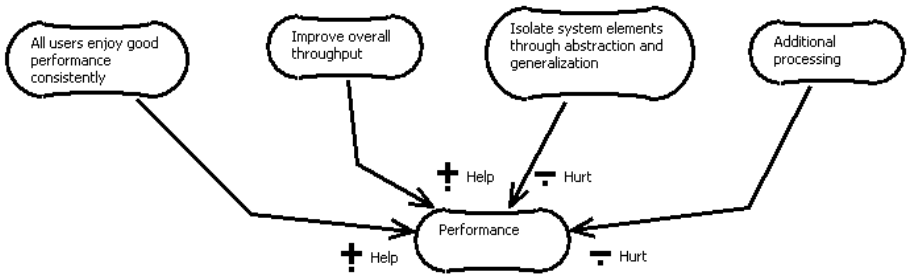


Figure 11. Forces of architectural design that impact performance

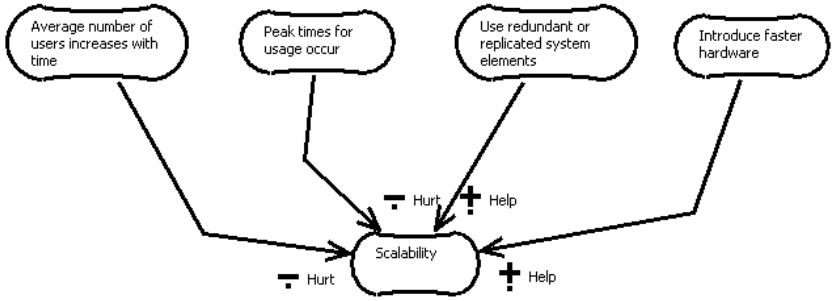


Figure 12. Forces of architectural design that impact scalability

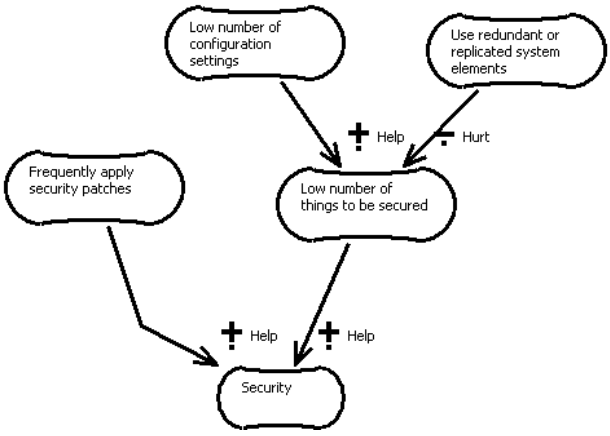


Figure 13. Forces of architectural design that impact security

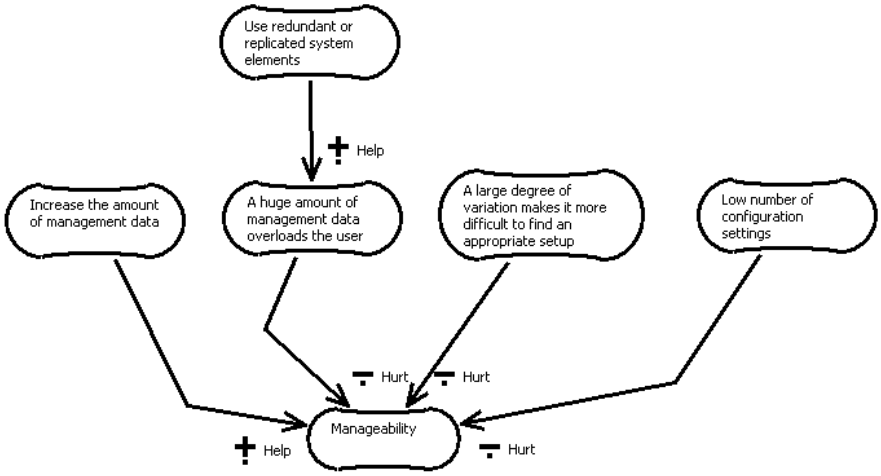


Figure 14. Forces of architectural design that impact manageability

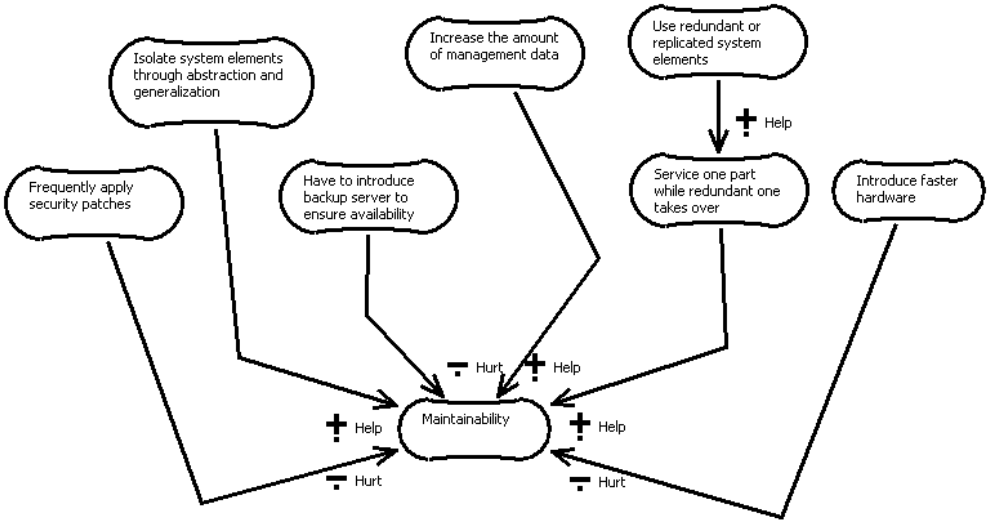


Figure 15. Forces of architectural design that impact maintainability

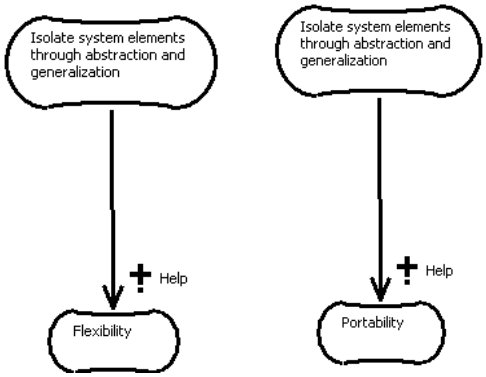


Figure 16. Forces of architectural design that impact flexibility and portability

As an example, we have applied our pattern formalization to one of the architectural views described in (Avgeriou and Zdun, 2005). The Layered View is concerned with how interacting parts of a system remain decoupled. Two patterns are classified under the Layered View: Layers and Indirection Layer (see Figure 17).

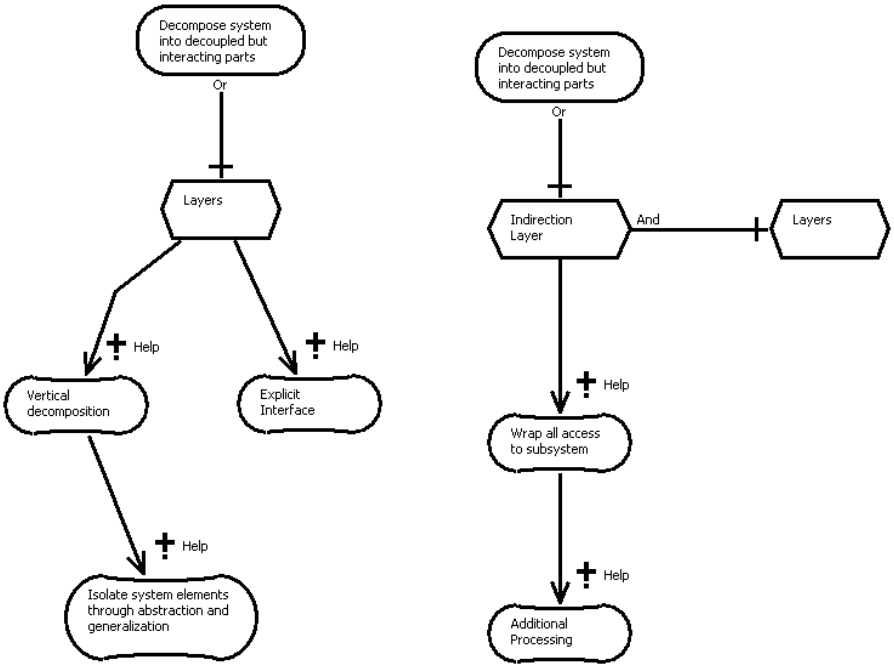


Figure 17. Layers (left) and Indirection Layer (right)

Figure 17 helps us conclude that the patterns impact performance more or less negatively and maintainability, flexibility, and portability positively because it references the softgoals “Isolate system elements through abstraction and generalization” and “Additional Processing” which also occur in Figure 11, Figure 15, and Figure 16.

Figure 10 to Figure 17 represent one complete view of the system in terms of forces and non-functional requirements. Other views, however, can be created if they are deemed useful by rearranging the forces and non-functional requirements. The relationship between the pattern “Layers” and the non-functional requirements performance, maintainability, flexibility, and portability is shown indirectly in Figure 17 with the help of the referenced softgoal “Isolate system elements through abstraction and generalization”. In this case, the organization of the GRL graphs favors reusability of the graphs in Figure 10 to Figure 16 over directly showing, on a single GRL graph, the impact of the

pattern “Layers” on non-functional requirements. The latter view, however, can be easily created from the presented GRL graphs.

DISCUSSION AND FUTURE TRENDS

This example demonstrates how GRL graphs capture pattern forces and can be used to assess the qualitative impact of various solutions to a functional goal, in context. The benefits of the proposed process become even more interesting as the system gets more complex and trade-offs are more difficult to assess due to numerous interrelated contributions and side-effects.

The proposed process for URN-based trade-off analysis is not limited to individual subgoals. It can easily be extended to pattern combinations addressing all subgoals at once, hence providing for global impact analysis and guidance, at the system level. This may go beyond the needs of designers and system architects who may only be interested in solving a focused design problem, but this level of evaluation is nevertheless possible.

Obviously, in addition to having patterns formalized with URN, the more advanced benefits require additional investments that not all modelers may be willing to make: a system-level GRL model for assessing the global impact of selected patterns.

There is also a general trend towards formalizing aspects of patterns, whether they are related to the relationships between patterns, or to properties of individual patterns. Often, these efforts have been confined to certain domains, which are more amenable to formalization, such as the area of security patterns. One example is the formalization of security properties that are satisfied by application of a pattern (Wassermann and Cheng, 2003), another is the formalization of security pattern properties in (Mouratidis et al., 2005), which allows the authors of a pattern language to assess the completeness of the language. In this work, the pattern solutions are modeled in agent-oriented models in the Tropos modeling framework. A formal language for Tropos models is used to formalize the problems, solutions, and consequences of these patterns. These properties are expressed in logic statements over the components of the solutions. Some of this logic-based formalization could be added to GRL, although it does not appear essential in our current trade-off analysis context.

Aspects, which are concerns that crosscut dominant decompositions, represent another trend identifiable in the software community in general. Aspects have been studied for a decade and are used to compensate several weaknesses of object-oriented programming languages (e.g., scattering and tangling of functionalities and other concerns). Existing design patterns have been recast for new aspect-oriented languages, e.g. see (Hannemann and Kiczales, 2002), whereas new patterns specific to such languages have started to appear. More recently, aspect-oriented concepts have been included in modeling languages, closer to design and requirements. For instance, (Jacobson and Ng, 2004) present an approach where aspects are derived from UML use cases, whereas (Yu et al., 2004) present an approach where aspects are inferred from goal models. The impact of the availability of such enhanced modeling languages requires further exploration. For instance, we believe aspect-oriented concepts can easily be added to URN. This could help close the gap between aspect-oriented modeling and programming languages and at the same time open the door to new types of patterns that are more abstract or closer to requirements than the current generation of design patterns.

CONCLUSION

In this paper, we have presented an approach where architectural patterns are formalized with the Goal-oriented Requirements Language (GRL). Our main objective is to describe patterns in a way that the various and conflicting forces involved can guide, in a given context, the selection of the most suitable patterns or combinations of patterns amongst many alternatives.

Forces and contributions for individual patterns are captured using GRL. Combinations and side effects (correlations) are described with AND graphs, and alternative combinations for a given (functional) goal are represented with an OR graph. With the help of strategies (i.e. initial selections of candidate patterns) and propagation rules, designers can assess the impact of their selection on the forces and find a suitable solution in their context. This context can itself be modeled with GRL, first at the actor/dependency level and then at the level of intentional elements (goals, softgoals, tasks, etc.) for the system. This enables global and rigorous assessments to be made, even when many functional sub-goals are considered.

To take full advantage of URN-based formalization of design patterns, a process was briefly introduced and illustrated with a case study where many combinations of patterns could be used to achieve the same functionality while leading to different trade-offs involving non-functional aspects such as maintainability and performance. A prototype Eclipse plug-in, which was used to create and evaluate the URN models presented here, already exists to support such process, and is still evolving to support new types of analyses and transformations (Roy et al., 2006).

We believe this formalization approach will provide means to get rapid and global trade-off analysis results in context and make better use of current and future design patterns.

REFERENCES

Alexander, C. (1979). *A Pattern Language*. Oxford University Press.

Amyot, D. (2003). Introduction to the User Requirements Notation: Learning by Example. *Computer Networks*, 42(3), 285-301.

Araujo, I., and Weiss, M. (2002). Linking Non-Functional Requirements and Patterns. *Conference on Pattern Languages of Programs (PLoP).* Electronic Proceedings, jerry.cs.uiuc.edu/~plop/plop2002 (last accessed Nov. 2005).

Avgeriou, P., and Zdun, U., (2005). Architectural Patterns Revisited - A Pattern Language. *Tenth European Conference on Pattern Languages of Programs (EuroPlop)*, Irsee, Germany, July.

Chung, L., Nixon, B., Yu, E., and Mylopoulos, J. (2000). *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers.

Chung, L., Supakkul, S., and Yu, A. (2002). Good Software Architecting: Goals, Objects, and Patterns. *Information, Computing & Communication Technology Symposium (ICCT- 2002), UKC'02*. Seoul, Korea, July 8-11.

Coplien, J. (1996). *Software Patterns.* SIGS. Available electronically: http://users.rcn.com/jcoplien/Patterns/WhitePaper/SoftwarePatterns.pdf (last accessed March 2006).

Dyson, P. and Longshaw, A. (2004). *Architecting Enterprise Solutions – Patterns for High-Capability Internet-Based Systems*. John Wiley & Sons, Ltd, 364 pages.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Gross, D., and Yu, E. (2001) From Non-Functional Requirements to Design through Patterns. *Requirements Engineering*, 6(1), 18-36, Springer.

Hannemann, J. and Kiczales, G. (2002) Design Pattern Implementation in Java and AspectJ*, 17th OOPSLA*, November, 161-173.

ITU-T (2003). *Recommendation Z.150 (02/03), User Requirements Notation (URN) – Language Requirements and Framework.* Geneva, Switzerland.

Jacobson, I. and Ng, P.-W. (2004). *Aspect-Oriented Software Development with Use Cases*, Addison Wesley Professional.

Mouratidis, H., Weiss, M., and Giorgini, P. (2005). Security Patterns Meet Agent Oriented Software Engineering: A Complementary Solution for Developing Secure Information Systems. *Conceptual Modeling (ER)*. LNCS 3716, 225-240, Springer.

Ong, H., Weiss, M., and Araujo, I. (2003). Rewriting a Pattern Language to Make it More Expressive. *Hot Topic on the Expressiveness of Pattern Languages, ChiliPLoP*. Carefree, USA, March.

Rising, L., (2000). *Pattern Almanac 2000*. Addison-Wesley.

Roy, J.-F., Kealey, J. and Amyot, D. (2006). Towards Integrated Tool Support for the User Requirements Notation. To appear in *SAM'2006*, Kaiserslautern, Germany.

Taibi, T. and Ngo, D.C.L. (2001). Why and How Should Patterns Be Formalized. *Journal of Object-Oriented Programming (JOOP)*, vol. 14, no 4, 8-9.

URN Focus Group (2003a). *Draft Rec. Z.151 – Goal-oriented Requirement Language (GRL).* Geneva, Switzerland, Sept. 2003.

URN Focus Group (2003b). *Draft Rec. Z.152 – Use Case Map Notation (UCM).* Geneva, Switzerland, Sept. 2003.

Wassermann, R., and Cheng, B. (2003). *Security Patterns*. Technical Report, MSU-CSE-03-23, Michigan State University.

Weiss, M. and Amyot, D. (2005). Business Process Modeling with URN. *International Journal of E-Business Research*, 1(3), 63-90, July-September.

Yu, E. (1997). Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*. Jan. 6-8, Washington D.C., USA, 226-235.

Yu, Y., Leite, J.C.S.P., and Mylopoulos, J. (2004). From Goals to Aspects: Discovering Aspects from Requirements Goal Models. *12th IEEE Int. Conf. on Requirements Engineering (RE 2004)*, Kyoto, Japan, September, 38-47.

Using Patterns to Create a Service-Oriented Component Middleware

A Pattern Story

James Siddle
www.jamessiddle.net
jim@jamessiddle.net

Intended Audience

This paper is a pattern story; a description and discussion of the patterns that were applied on a particular project, written after the fact. The paper includes descriptions of the ways in which a number of patterns were applied, the degree of success achieved in applying those patterns, and a discussion of interactions and relationships that were observed between the patterns. This paper is expected to be relevant for pattern users, who may learn some of the benefits or pitfalls associated with the specific patterns discussed and with patterns in general; for pattern writers, who may learn of new examples, interactions, or relationships associated with the patterns; and for pattern theorists who may be able use the following material to make observations about the process of applying patterns in general. For more information on pattern stories, including a definition, see Kevlin Henney's "Context Encapsulation: Three Stories, a Language, and Some Sequences" [Henn06].

Introduction

This paper describes the patterns that were applied during the design of a service-oriented component middleware for telephony devices with limited resources. The problem context of the project where the patterns were applied is described first, stated as a set of *forces,* and this is followed by a brief overview of the associated architecture vision. The patterns that were applied to solve the problem and create the software architecture are then described, along with their method of application and relationships to one another. Finally, an overview of how patterns resolved the forces stated in the problem context is given.

This paper focuses on the concrete experiences of applying patterns on a particular project, rather than attempting to describe an idealised approach to applying patterns to solve a problem. It is assumed that the reader is familiar with the patterns that are discussed.

In the context of this paper the following definition of the term "Service Oriented Architecture" taken from Wikipedia [WIKSOA] is used:

> *"the term **Service-Oriented Architecture** (SOA) expresses a business-driven approach to software architecture that supports integrating the business as a set of linked, repeatable business tasks, or "services". Services are self-contained, reusable software modules with well-defined interfaces and are independent of applications and the computing platforms on which they run."*

The above definition captures the concept of SOA as it was understood by the team working on the project being discussed. The key characteristics of SOA that were important for the project were those of self-contained reusable software modules, well-defined interfaces, and platform independence, hence the use of the term "service-oriented component middleware" to describe the system being discussed. The platform independence in this case was required to enable reuse of code on alternative platforms.

Problem Context

The project where the patterns were applied, and that prompted the writing of this paper, was the software development project for a range of commercial telephony devices with limited resources. One of the main goals of the project was the creation of a new software architecture and service-oriented software components that could be reused on future software development projects within the organisation. That goal is the focus of this paper; it was the main driver behind the application of patterns on the project, and led to a particular emphasis on architecture development during the early project phases. The other goals of the project are not described here for confidentiality reasons.

Several reusable, telephony-domain specific "services" had been identified during the inception of the project, and the software architecture under development was specifically required to support those services. Typical examples of the required services are:

- a service to allow telephone directory lookups both locally within the user's personal directory, and remotely in enterprise directory applications;
- a service to provide "buddy" presence information to local telephony applications, and to propagate user presence information to remote enterprise applications;
- and a service to record and manage telephony-related user actions and associated events for subsequent user and administrative reference.

There were several reasons behind the above goals – these are stated below as *forces* which define the problem context:

- *Future reuse of software components in a product-line development strategy* – the development of future software systems based on a set of reusable components was a key goal.
- *Seamless (re)deployment of services at runtime* – a dynamic approach to service deployment and re-deployment was required in order to allow flexible management and updates of telephony device software.
- *Service invocation transparency independent of deployment model* – on the project in question, it was recognised that the ability to invoke services both locally and remotely in a transparent way was required, as was a decoupling between service interfaces and providers. This was to enable flexible composition of telephony services into new configurations for new development scenarios.
- *A common approach to management, testing etc of services* – it was desired that there was a simple and consistent way of managing, testing, and other general purpose handling of services within the target system.
- *The reuse and protection of existing software components* – a large base of existing telephony software was available when the project commenced, and making the optimum use of that software was desired.
- *Acceptable performance within resource-limited environments* – the software architecture in question was being targeted at telephony devices with limited resources such as memory, processor power, and non-volatile storage. The expectation was that the resources would be not be upgradeable.
- *Independence from specific computing platforms* – the software architecture and reusable software components were required to have no dependencies on particular computing platforms, given the product line development strategy.

Project Background

The development process that was applied during the initial phases of the project was an agile process based mainly on Scrum [ScBe01] and Extreme Programming [BeAn04]. The selected process was user-story driven, and was also architecture-centric hence the particular focus on the application of architectural patterns, concurrency and optimization patterns, and resource management patterns as described in the *"Pattern Oriented Software Architecture"* books (see [POSA1-3]).

Generally speaking, candidate patterns were investigated and selected at the start of each iteration, based on the requirements that served as the input to that iteration. In addition to the requirements list, a high level architecture vision was established to drive the pattern selection. A team architect guided the team through pattern selection and subsequent rough upfront design, and the patterns were applied as the iteration progressed using Test-Driven Development and Continuous Integration. The resulting implementations were proven at the end of the iteration according to the acceptance tests, which also served as input to the iteration.

The initial team structure was based around the architectural building blocks described in the following section. It should be noted that the patterns described in this paper were applied predominantly by one team, who were responsible for creating the service-oriented component middleware and initial services.

The programming language selected for the majority of development was C++.

Key Terms / Architecture Vision

The software architecture vision required the creation of a service-oriented component middleware for telephony devices with limited resources. The following table outlines the key concepts that were introduced at the start of the project, and is followed by an diagrammatic overview of the envisaged architecture:

Service	Reusable software asset encapsulating business logic, typically composed of smaller granularity components such as classes.
Service Container	Manages service lifecycle, communication, and invocation enables services to focus on business logic.
Service Interfaces	The interfaces exposed by a particular service, representing the capabilities that service clients can invoke. Expected to support both local and remote invocation.
Service Bus	General purpose service communication mechanism providing location transparency and encapsulation of underlying communication.
Platform Abstraction	A set of interfaces providing platform independence.

The diagram shown in Figure 1 gives a broad overview of how several of the concepts described above related to one another in the architecture vision.

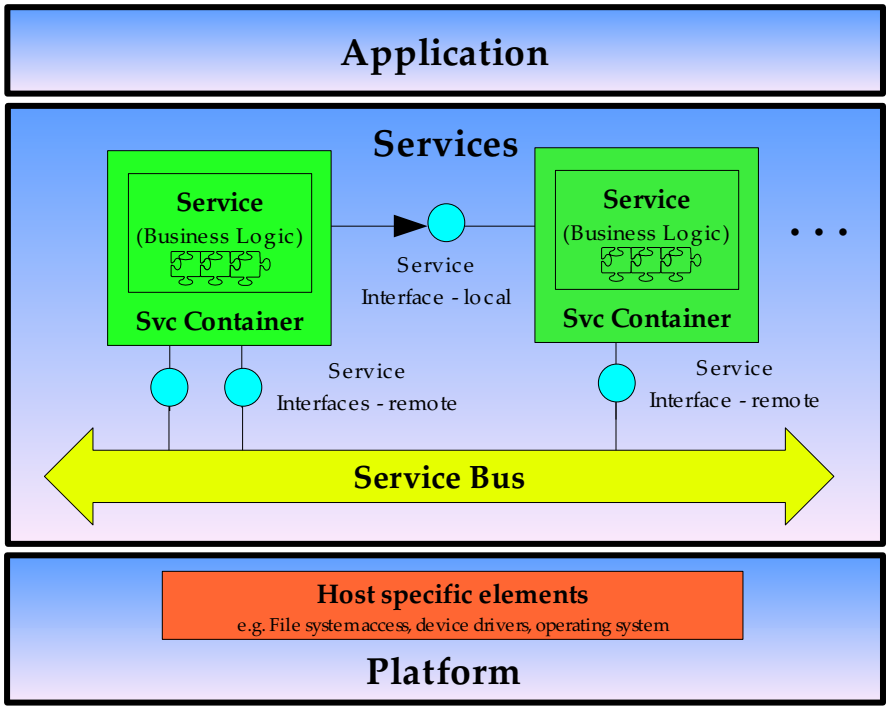


Figure 1: Architecture Vision Overview

Solution

This section describes the patterns applied on the project, the reason for their selection, and discussion about the pattern applications, and the interactions or relationships that were observed between them.

Establish Layers

The **layers** [POSA1] pattern gave the overall, fundamental structure to the software architecture in order to support the project's product-line development strategy. This pattern was part of the architecture vision as shown above, so was not selected and applied as described in the 'Project Background', rather it framed the whole project team's thinking from day one.

The layers were selected such that the high-value, reusable, service and service-related code was localised in the *Services* layer, and could only make use of core platform functionality via the strict interface to the *Platform* layer. The *Application* layer localised code was, along with the *Platform* layer, expected to be specific to each computing platform.

The impact of this pattern on the rest of the project was profound – every following implementation decision was framed by the established layers. Unlike most of the following patterns, there was no actual code associated with the pattern – rather it formed the basic conceptual framework for all following implementation.

There were however two concrete artifacts associated with the layers pattern: the initial team structure, and the initial configuration management data-store structures. Both of these were based around the layers that were established, which is felt to have given the layering greater substance and meaning on the project. This was a logical approach given the product-line development strategy where different teams can be expected to be created and disbanded based around different architectural blocks over time, and where different blocks are likely to be accessed, modified and used in different ways throughout the lifetime of the software. *"Conway's Law"* [Conway68] and the related organisational patterns such as *"Organisation Follows Architecture"* [Coplien95] provide deeper insight into how team structure is related to architecture.

It is also interesting to note that the layers pattern was actually applied a second time, later on in the project. The reason for this was that certain other architectural abstractions were observed within the previously established layers, also the configuration management data-stores – that were based around the original layering - were found to be too large, which caused technical problems with the infrastructure. As such, the *Services* layer was decomposed into (from top to bottom) *Business Services*, *Infrastructure Services*, and *Framework*. None of the layering, except

for the original *Platform* layer, was strict - rather it was based around logical, observable groupings of software elements. Similar decomposition took place within the *Application* layer, which resulted in *Application*, *App Framework*, and *Presentation* layers.

The second application of layers in fact acted as a stepping-stone to a final, UML "package" based approach. That is, the layers introduced by the second application of the pattern were subsequently transformed into packages with explicitly defined dependencies. This was to enable more accurate representation and analysis of the dependencies between the layers.

It should be noted that the introduction of layers was not perceived to have introduced any particular performance problems on the project, however this is because the targeted device hardware had the capacity to deal with the associated overhead. This may not be the case in all scenarios; care should be taken when applying this pattern where the resulting software has to run in environments with limited resources.

In the discussion that follows, the patterns were applied predominantly in the context of the *Framework, Infrastructure Services,* and *Business Services* layers, except for **wrapper-facade** which was applied in the context of the *Platform* layer.

Abstract the Platform

The next step was to introduce platform independence. This is because without platform independence, it would be difficult to do anything without violating a core principle of the architecture vision. A **wrapper-facade** [POSA2] approach was taken to encapsulate low-level, host-specific functions and data structures, as well as calls to legacy and third party code.

By applying the wrapper-facade pattern it was possible to define 'pure' OS abstractions that would be realised for particular Operating Systems as needed by telephony product developments. Additionally, several legacy software components were integrated into the project that were known to have maintainability issues. The creation of wrapper-facades for these components not only hid the details of complex interactions between the legacy components, but also provided an abstraction that protected new code from legacy code. A similar approach was taken for certain third party software components.

A typical example of legacy code contained by a wrapper-facade was a group of related modules responsible for interacting with a remote telephony-device management platform. Operating system examples include file access and network access.

By abstracting operating system, legacy, and third party code the fundamental bedrock of platform independence was laid. One consequence of this approach was a

requirement for support and development of the abstractions. Given the iterative and incremental nature of the project, abstractions were only added as needed, however this meant an ongoing investment of time and effort to create the abstractions, as well as on some occasions reworking of code written before abstractions were available.

It is thought that by introducing the concept of platform abstraction at the very start it was much easier to establish momentum in the creation and maintenance of the abstractions throughout the project.

Manage Service Lifecycle

With platform independence established, it was necessary to introduce the concept of services into the system, a reasonable place to start being the creation and deletion of services. **Component configurator** [POSA2] was chosen in the early phases of the project to manage service lifecycle, and it was applied largely as described in the pattern.

The initial implementation of the component configurator pattern provided a service creation and initialisation mechanism in addition to laying the groundwork for the re-initialisation and shutdown of services in future iterations. Typical examples of initialisation performed by services when prompted by the component configurator implementation include:

- service discovery;
- subscription to other services for event notification;
- and the creation and initialisation of platform and third party resources such as network sockets or telecommunication protocol stacks.

The component configurator implementation's role was later expanded to provide for the discovery and the management of the connections between services. One of the architectural goals of the project was the decoupling of services from underlying communication details such as transport, message protocols, location etc, and the implementation was extended to provide a dynamic lookup mechanism that could be called by services to discover other services.

The implementation of the component configurator pattern partially delegated this “discovery” responsibility to another element within the system, but on reflection it was not appropriate to assign the responsibility in this area at all. The management of connections between services should have been handled separately, but because of its central role in the system it was easy to modify the responsibility of the component configurator implementation. This made the associated classes more difficult to understand and on reflection it would have been better to create a separate class with the responsibility of managing service connections, which could consult the component configurator implementation if necessary.

Fortunately it was possible to limit the effect of the inappropriately assigned responsibilities to a small number of classes by applying **encapsulated context object** and **decoupled context interface** (see [Henn06]) to the component configurator implementation to selectively expose the discovery methods. The component configurator implementation was modified to implement a "Service Context" interface that was passed to services via their initialisation method. Similarly, a "Framework Context" interface was implemented and passed to several different framework layer elements to expose related methods.

On reflection, both the naming and the definition of the "Framework Context" interface was too broad, and it was passed to too many framework layer elements. It would have been better to break this context interface down, rather than having a 'catch all' interface for framework layer elements.

Introduce Service Communication, Introduce Location Transparency

Following the abstraction of the platform, and the creation of a basic service lifecycle management mechanism, the next problem was that services needed to communicate with one another and also that one service should not be aware of the location of another service.

Both of these problems were dealt with in one go. A "Service Bus" subsystem was introduced to encapsulate communication between named end-points. This was seen as being a key step in achieving communication between services, which were initially able to create instances of named Service Bus end-points for communication with other services. This approach was inspired by the Qt "CopChannel" mechanism (© TrollTech - see [TTECH]) and the **broker** pattern [POSA1].

The initial implementation of the broker pattern provided intra- and inter- process communication. One consequence of this was that services could be easily deployed into different processes via the component configurator implementation, which later proved useful in separating services according to how critical their operation was. For example a service providing essential telephony event processing would be in a different process to a service collecting non-essential performance statistics.

The broker implementation was later extended by adding a "Bridge" which enabled remote communication. The initial design for this element of the architecture was not directly inspired by the broker pattern but was thought of independently, it was only later in the project that the correlation between the bridge role and the element in question was identified. The cohesion of the bridge role into the rest of the system is thought to have been adversely affected by this sequence of events, and it is thought that a more detailed understanding of the broker pattern across the project team could have alleviated this.

Examples of remote applications that invoked services via the bridge include a test tool, and a telephony device management application. Outgoing service invocation via the bridge was not elaborated on this particular project.

Establish Service Interactions

In the system that had been created so far, services were able to talk to one another, but the nature of the interactions between services was not understood.

Client-server interactions (see [MODPA]) were established as being central to the overall behaviour of the system – it was realised that services were expected to be both clients and servers. Other elements of the system such as application presentation logic were only expected to be clients.

Two types of client-server interaction were introduced: *Synchronous* and *Asynchronous*. The Service Bus subsystem automatically provided for both types of interaction given that it was purely message based, and supported a read method which could wait for an incoming message.

A typical example of a synchronous interaction would be a subscription request from one service to another, where the requesting service required a successful response message before proceeding; subsequent event notification is a typical example of asynchronous interaction.

In addition to establishing interaction patterns between services, the application of client-server also supported separation of services into easy to understand client and server roles.

Once asynchronous interactions were established between clients and servers, the **asynchronous completion token** [POSA2] pattern was applied to allow clients to deal effectively and efficiently with asynchronous responses. It is worth noting that an initial attempt was made to apply this pattern to the system infrastructure, however this was activity was not continued because it was thought at the time that the pattern was more applicable to service interactions directly, rather than via supporting infrastructure. On reflection it may have made sense to complete this activity in order to provide general purpose support for the pattern within the architecture - the pattern was not easily understood by service developers, so was only applied to a relatively small number of service interactions.

Abstract Execution

Now that services were able to communicate and interact with one another, it was noticed that services were handling their own execution and were running their own message loops to process messages received over the Service Bus. The architecture vision indicated that services should be focused on business logic rather than execution details, so abstracting service execution was an essential next step.

The **executor** [Crah02] pattern was applied to achieve this. Given that all interactions between services would be conducted via the Service Bus introduced above, it was recognised that an executor could be created that took queued messages off the Service Bus, determined execution context for a service based on the received message, then passed the message to the service for handling.

Initially, a single-threaded executor was created, however this was quickly modified to introduce a Thread class, and a Thread Pool via **pooling** [POSA3], which gave the executor the ability to support concurrent execution of services. Further modifications provided configurability, whereby it was possible to select either a single or multi-threaded execution model for each service via a configuration file.

Practically speaking, the threading configuration mechanism was not extensively used because the vast majority of services were required to handle multiple simultaneous requests to ensure acceptable system responsiveness.

Provide Well-Defined Interfaces

Once service communication, interaction, and execution abstraction had been established, the next key step was to put in place well defined interfaces between services. This was because the system that had been put in place so far required services to process the messages they received themselves. Keeping services focused on business logic rather than communication details such as message format was a key aspect of the envisaged architecture. The **explicit interface** [BusHen] pattern was applied to achieve support for well-defined interfaces between services, and the **proxy** [GoF] pattern was applied to achieve decoupling of business logic from communication details.

Clients that required the capabilities described by a particular interface would discover a service that supported that interface using the "Service Context", then make method calls against it. Service interfaces were given names in order to uniquely identify them. For services that could be resolved locally, the actual Service object would be called via the interface whereas for remote services the proxy object would be called. It should be noted that the 'local' invocation mechanism was quickly disabled because it bypassed the “abstracted execution” described above, however it was recognised that under certain circumstances it could provide a useful way of optimizing service invocation for performance reasons, especially if it could be enabled via configuration.

The explicit interfaces were defined in pure abstract base-classes to enable both proxy and service classes to easily implement them, and also to allow seamless transition between local and remote invocation. Each service or proxy that supported an interface was required to inherit from the abstract base class that defined the interface. In order for services to support multiple interfaces, a Java-style approach to multiple inheritance was taken, i.e., via multiple, purely abstract base classes, each of

which defined an interface that the service would support.

It was realised that the use of multiple inheritance in C++ could lead to the problem where there are multiple copies of a base class for a particular multiply-inheriting class. However, the fact that multiple inheritance was only used for interface implementation was thought to alleviate this problem somewhat.

Typical operations found on service interfaces include subscription to telephony related events; actions such as making a call, or changing presence status; and requests to access or modify configuration values or data variables.

Subsequently, this approach was found to conflict with the platform independence goals of the project. This is because each explicit interface was defined in an abstract C++ base class, which is inappropriate for service clients that do not share the same underlying platform, e.g. a Java-based client on a remote system. For service clients on other platforms, it was necessary to provide an alternative definition of the interface, which in this case was in the form of documentation of underlying message formats. On reflection it would have been more appropriate to provide the interface definition in a platform independent form, for example via an "Interface Definition Language" such as CORBA IDL or WSDL.

Symmetrical **invoker** counterparts to proxy objects, which were inspired by **command** [GoF] but are described formally as a pattern in Remoting Patterns [KiZdVo], were then introduced to invoke the explicit interfaces on remote services. A one-to-one relationship was established between proxy and invoker objects, with the explicit interface being the common association.

The executor class was modified such that for each message received by a particular service, a suitable invoker class was discovered via the "Framework Context". The executor delegated responsibility for invoking the service to the invoker object, passing it the message as well as a reference to the service that was being invoked and that supported the required interface.

Both proxy and invoker components required the support of the "Framework Context" object to establish certain communication details such as the Service Bus locations for return messages.

One consequence of using the explicit interface, proxy and invoker patterns that became apparent as the project progressed was that the complexity of service interaction was difficult to understand for developers who were relatively new to Patterns and Object-Oriented development. Also establishing relatively simple service interactions required a significant amount of development. On reflection, it would have been better if the proxy and invoker classes had been automatically generated – in fact a partial "Service Generator" was written that generated skeleton code for proxy, invoker, and service classes from an explicit interface. Unfortunately due to project time pressures it was not possible to develop this further.

Observer [GoF] was then applied to support notification of events asynchronously to multiple subscribers via well-defined interfaces. The application of this pattern built upon the asynchronous behaviour introduced by client-server, as well as the interfaces introduced by explicit interface. Notification events were communicated to observers via call-back explicit interfaces and their respective proxy and invoker objects.

Incidentally, observer was used widely in the software system. It was applied to solve a number of different problems and, along with the proxy pattern, became a core project metaphor that was often discussed and referenced in design sessions. Problems such as how to notify multiple subscribers of physical device events, managed-data update events, and other telephony related events were solved by applying this pattern. Observer was the key pattern for handling notification of events within the architecture.

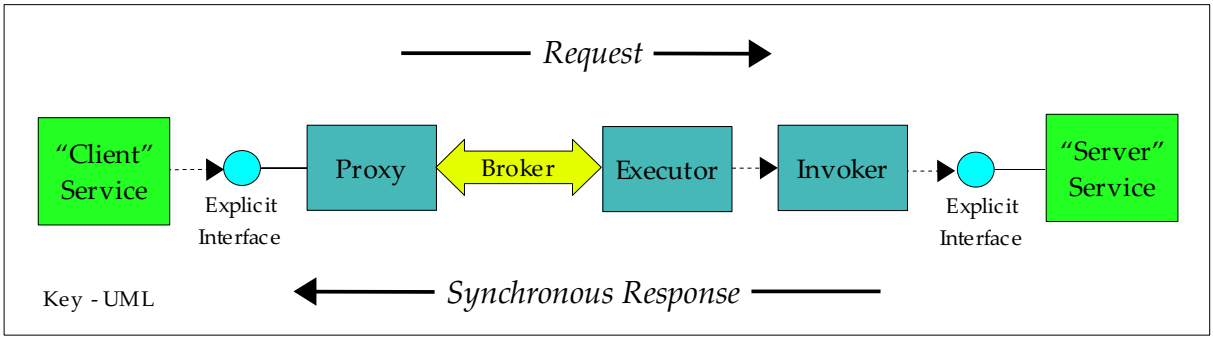


Figure 2: Service invocation

Ensure Service Transparency

With well-defined interfaces in place between services, there was still the problem of ensuring that there were no dependencies between the clients of particular interfaces, and the services which implemented those interfaces. Service transparency was required to solve this problem.

The "Service Context" provided both local and remote discovery of services supporting required explicit interfaces. **Lookup** [POSA3] was applied to achieve this. Two sources were consulted during discovery – the local "repository" of services which was managed by the component configurator implementation, and a "Service Registry" in a well known location.

If a service supporting the required explicit interface was found in the local repository, a direct reference to the service object would be returned. If no service supporting the interface was found locally, or if local discovery was disabled as described in "Introduce Well-defined Interfaces" above, then the registry of services was consulted and a proxy was created based on the discovered information. A Service-Bus address representing the location of the service that supported the

required interface was passed to the proxy object, along with a Service-Bus object so that it could send messages. A similar lookup mechanism was established to enable the executor implementation to lookup invoker objects based on the interface being invoked.

This approach ensured that service clients had no dependencies on the concrete services that implemented the explicit interfaces they invoked. It also helped to ensure that proxy and invoker objects were not associated with particular remote implementations of explicit interfaces, rather they simply provided a way of transforming invocations on interfaces to and from a message-based form.

One significant consequence of this approach subsequently became apparent. The interaction between the explicit interface, proxy, invoker, and component configurator implementations relied heavily on C++ Run-Time Type Information (RTTI). This was because dynamic casting was required when casting objects created in different different shared libraries or executable files to specific explicit interface types, such as when a client service cast a proxy object to a particular interface in order to make service requests.

This requirement on RTTI was recognised as being a restricting factor in the environments with limited resources that were being targeted on the project. On reflection it may have made more sense to consider approaches which did not require such extensive use of RTTI in order to avoid this consequence. Sacrificing the dynamic deployment provided by component configurator, and relying on a single software bind would have been one such approach.

The initial service discovery mechanism was based purely on required interfaces, however it was recognised that in the future it would be possible to extend available discovery options via the “Service Context”. For example, discovering services by criteria such as “number of supported database entries” or “security level” could be supported.

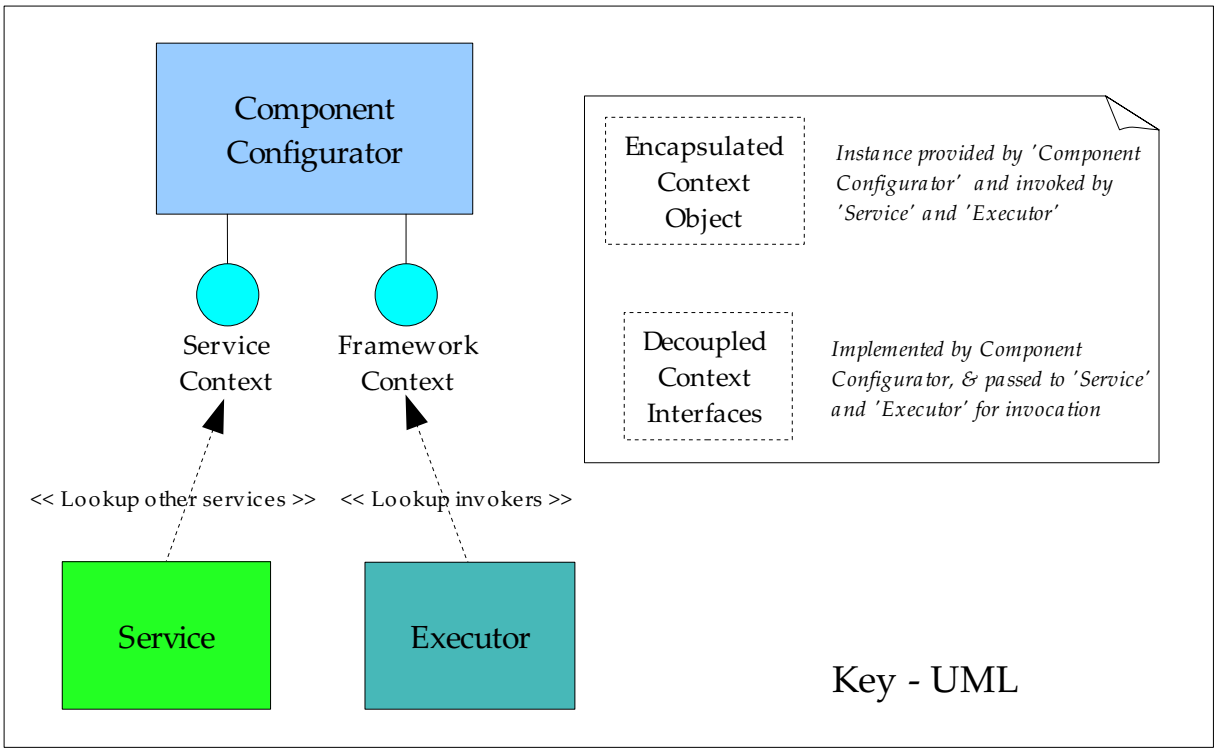


Figure 3: Contexts provided by Component Configurator implementation

Provide for Extensibility

The core framework of the software system was now in place – managed services could interact with each other both synchronously and asynchronously over a service bus supporting location independence, were unaware of communication and execution details, and were abstracted from service provider details via well-defined explicit interfaces, and proxy and invoker objects.

The **interceptor** pattern [POSA2] was then applied to achieve a core project goal of extensibility. It had been pre-selected as a pattern to be applied in the overall architecture vision. The goal was to introduce an interception point within the 'Service Bus' – the core framework element that was responsible for transporting messages between services and service clients.

In order to achieve a degree of consistency between different interception points, it was decided to create a set of base classes that could be sub-classed for each concrete interception point. **Template method** [GoF] was applied to establish a common mechanism for dispatching events to interceptors, where certain concrete implementation decisions were delegated to the subclasses associated with particular interception points.

It was thought that the base classes would prove useful at a later point in the project when dynamic management of interceptors was envisaged, potentially via a

component configurator. On reflection however, the introduction of these classes without an explicit requirement for them may have been unnecessary at the time the classes were introduced. Keeping the initial implementation as simple as possible, and introducing an abstraction as and when it was needed may have been more appropriate.

The Service Bus underpinned the entire software base, so the interception point was made generic to ensure maximum adaptability and applicability, though it was felt that this may have performance implications, especially if many concrete interceptors were plugged in. However the requirements for the interception point were modest in the first usage scenario, and as of the current date no performance problems have been attributed to it. Careful selection of interceptors will be required to ensure performance remains acceptable in the future, with performance profiling and code efficiency enhancements potentially being necessary in the worst case scenario.

A proof of concept 'logging' interceptor was initially created to intercept and log messages being transmitted via the Service Bus, and a further interception point was later introduced at the point of service invocation in the executor implementation. A performance-recording interceptor that recorded performance statistics for each service invocation was written to make use of this interception point. Security related interceptors are expected to be created in the future.

The “Bridge” role from the **broker** pattern was then implemented using the **template method** pattern in order to support remote service invocation via a number of different underlying communication technologies. The problem of how to invoke services remotely was recognised early in the project. Services could and would be invoked via a number of different remote transport mechanisms and technologies, UDP/IP and USB being two examples. A general purpose approach was considered necessary to prevent complication and inconsistency being introduced into the system, and to support the platform independence goal.

A template class was introduced which understood how to interact with the Service Bus 'locally', and which delegated the implementation details of interacting with the 'outside world' to sub-classes via template methods.

This approach provided an easy way of adding new sub-classes for different types of external communication at later stages in the project, and also enabled the separation between 'internal' and 'external' service communication details. One consequence that emerged later in the project was that the implemented template class was not ideally suited to all types of communication technology. The initial implementation was based around the first required technology, but other technologies that were subsequently introduced did not fit perfectly with the template. On reflection it would have made more sense to establish a better understanding of the expected communication technologies in advance, and to define the template in a more generic way based around that understanding.

Ensure Acceptable Performance

With the core of the system plus a number of extensibility mechanisms in place, it was necessary to ensure that the system performed acceptably. A number of patterns were applied to achieve this, however it should be noted that the patterns were not applied as an afterthought – rather they were introduced as needed during development. They are gathered here for clarity, and to illustrate their relationship with the other patterns previously applied.

Pooling [POSA3] was applied in the executor to achieve efficient use of thread resources, and the application of **asynchronous completion token** [POSA2] was also recognised as supporting the efficient use of threads, i.e. services waiting for asynchronous responses could release threads back into the thread pool.

The **caching** pattern [POSA3] was employed to store proxy and invoker objects for future use. A central cache of proxies and invokers could be queried via the "Framework" and "Service" contexts. It was recognised that caching could also have been employed within the executor, for example the most recent invoker object associated with a particular service could be cached.

The local resolution of services supported by the **explicit interface** and **proxy** patterns as described in "Provide Well-Defined Interfaces" above was also recognised as contributing towards acceptable performance, by selectively bypassing communication and execution infrastructure.

Finally **lazy acquisition** [POSA3] was applied to the initialisation of services. This was achieved by exposing a startup method via the "Framework Context" interface. The component configurator implementation was modified to selectively initialise services based on a configuration parameter, and those services that were not automatically initialised at startup could be started via the context. The executor implementation was modified to initialise services the first time they received a message.

Apply Cross-Cutting Concerns

Having established the system core, extensibility mechanisms, and acceptable performance, the final step was to apply cross cutting concerns to the system.

The **singleton** [GoF] pattern was often dismissed during design sessions as being an "anti-pattern" – a "politically correct term for a global variable" as defined in the Wikipedia definition of the singleton design pattern, see [WIKSNG]. However, there were two closely related scenarios on the project where the singleton pattern proved useful – those of Logging and Exception Handling. Unfortunately Aspect Oriented Programming was not considered, in hindsight it may have provided a better solution, however a C++ tool enabling Aspect Oriented Programming was not immediately obvious so it was not considered.

One of the goals of the project was the reuse of existing software components, and it was thought that the handling of certain cross-cutting concerns within the software architecture could only be provided by some form of global instance. This is because several large software modules from previous projects were introduced into the system during development, and a common approach to cross cutting concerns was required for both existing and new code. Changing the existing code to introduce new mechanisms for cross cutting concerns was considered to be too much effort, so singleton 'logging handler' and 'exception publisher' classes were created.

To enable the reuse of existing code, calls to the logging singleton objects were wrapped in macros that were very similar to macros in the legacy code base. This allowed an easy transition to the new logging mechanism to take place within the existing code.

The singleton 'exception publisher' was responsible for ensuring that exceptions were reported to a single location within the system. This approach was selected to ensure that any exception within the system could be reported to an element of the architecture which was able to provide a higher level of system availability. This follows the “Fault Tolerant Hierarchy” approach described in chapter 6 of “Software Architecture In Practice” [BaClKa].

Although the use of singletons supported the reuse goals of the project, it was recognised that the use of this pattern caused all code to be dependent on particular instances of cross-cutting concern classes. On reflection it would have been better to allow multiple instances of logging and exception handling classes. Instances of these classes could have been provided to both service and framework elements via the “Service” and “Framework” context interfaces, while global instances would support cross-cutting concerns in the existing code. This would have provided a more flexible approach to applying cross-cutting concerns in the system while maintaining a degree of commonality between existing and new code, and would have avoided tying all code to particular instances of cross-cutting concern classes.

It is worth noting that the 'pure' singleton implementations were subsequently watered down to allow core infrastructure management code to dynamically manage the respective object instances.

Because the **executor** pattern encapsulated execution for all services, it was possible to apply a common exception handling strategy to all services. This was achieved by adding a general “catch all” block in the Thread class. In case of an exception, the catch block simply called the exception publisher then returned the executing thread back to the thread pool.

Forces Resolved

This section describes the way in which the selected patterns helped to solve the problem that was outlined at the start of this paper, by describing how each of the forces was resolved.

Future reuse of software components in a product-line development strategy

Layers provided the overall architectural structure required to support a product-line development strategy, by separating reusable and platform-specific software elements into different layers.

The use of **wrapper-facade** to encapsulate operating system, legacy, and third party code provided a way of separating existing code from new code via suitable abstractions, which supported reuse of both sets of code.

Management of service lifecycle via **component configurator,** and support for location transparency via **broker** and the Service Bus provided a way of deploying services in a wide variety of different scenarios, supporting reuse.

Client-server helped to establish clear roles within the system. This supported the creation of services with clearly defined roles and responsibilities, which contributed to the clarity and cohesiveness of the services and thus to the overall reusability.

Service-lifecycle was abstracted via **component configurator**, communication was abstracted via **broker** / the Service Bus along with **proxy** and **invoker**, execution was abstracted via **executor**, and cross-cutting concerns were abstracted via **singleton**. All of the above enabled services to focus on business logic, helping to create reusable assets.

Well defined, discoverable interfaces supported by **explicit interface**, **encapsulated context object**, and **lookup** also enabled services to focus on business logic because of the focus on the capabilities exposed by interfaces rather than implementation details.

Encapsulated context object provided a way of encapsulating the execution context of service and framework elements, and **decoupled context interface** provided a decoupling between context implementation and context user. The combination of these patterns provided coherent, decoupled access to execution context details.

Use of **observer** enabled the creation of reusable services that generated events for multiple subscribers, and that had few or no dependencies on other services. This resulted in greater reuse opportunities.

Provision of a generic **interceptor** within the Service Bus provided a useful extension point, while use of **template method** for implementing the Bridge

role of the broker pattern provided a mechanism for adding alternative types of external communication in the future.

Seamless (re)deployment of services at runtime

Component configurator provided a dynamic mechanism for managing service lifecycle and deployment. It provided a way of arbitrarily assigning services to run in different processes, and provided a way of quickly removing services that were discovered to be causing crashes or lockups. In some cases it was possible to fix defects simply by rearranging service configuration files. It also allowed for the creation of special configuration files for running tests.

Service invocation transparency independent of deployment model

Broker / Service Bus provided location transparency, while service transparency was provided by **explicit interface** and **lookup,** which provided decoupling between services and the interfaces between them.

The use of pure abstract C++ **explicit interface** classes worked against true service invocation transparency, because service clients on non C++ systems required alternative interface definitions.

A common approach to management, testing etc of services

The well defined, decoupled interfaces introduced by **explicit interface** enabled the creation of general purpose interfaces for management and testing, which services were able to support as required.

Singleton supported cross cutting concerns across a wide variety of software components, at different abstraction levels, with both new and legacy code.

The reuse and protection of existing software components

The use of **wrapper-facade** to wrap complex interactions between legacy code modules provided greater reuse opportunities, and also provided a layer of protection between new and legacy code, ensuring the two types of code did not inappropriately affect one another.

Explicit interface also enabled the reuse and protection of existing software components, by allowing them to be contained within services.

By providing **singleton** objects which supported interfaces required by legacy components it was possible to make more effective use of legacy components.

Acceptable performance within resource-limited environments

The **caching** of proxy and invoker objects supported efficient use of resources, along with the **pooling** of threads in the executor. **Asynchronous completion token** supported thread pooling by allowing services to release threads back into the pool while awaiting asynchronous responses.

By introducing well-defined interfaces via **explicit interface** it was possible to ensure quick and efficient access to services by providing local resolution, albeit at the cost of breaking execution encapsulation.

Unfortunately the interaction between the **explicit interface**, **proxy**, **invoker**, and **component configurator** patterns may overall have had a negative impact in terms of memory consumption and run-time performance due to reliance on C++ RTTI.

Lazy acquisition provided a quick and efficient service startup mechanism, by enabling service startup to be delayed until required.

Independence from specific computing platforms

Layers gave the fundamental architectural structure required to support platform independence, by separating reusable code and platform specific code into different layers.

The use of **wrapper-facade** allowed the definition of 'pure' OS abstractions which could be realised for particular Operating Systems, and thus allowed service and service-related code to be platform independent.

The use of pure abstract C++ **explicit interface** classes was recognised as working against platform independence, because non C++ service clients required alternative interface definitions.

Summary

This paper discusses how patterns were applied to realise a service-oriented component middleware for telephony devices with limited resources, with the key SOA characteristics of self-contained reusable software modules, well-defined interfaces, and platform independence.

Forces related to reusability, adaptability, and performance led to the application of a wide selection of patterns including **component configurator**, **interceptor**, **encapsulated context object**, and **proxy**. Patterns were central to the creation of a framework for achieving the project's architectural goals, though the use of C++ RTTI as required by the implementation of a number of patterns was recognised as potentially causing problems in resource limited environments, and the overhead required to develop the proxy and invoker classes due to the lack of a completed service generator was also a cause for concern .

On the project in question the entire service framework, namely the lifecycle management, communication, service interface, and cross-cutting concern elements to name but a few were developed in-house. There are advantages to developing such in-house solutions, however an alternative approach would have been to use a third-party framework such as embedded CORBA or Rational Rose RT. A number of these were considered early in the project, but were rejected as no suitable third party framework could be identified for the project.

It is also worth noting that the patterns applied on the project were not drawn from one particular *pattern language* - they were selected from a number of the most popular patterns publications, as well as being suggested in several cases by a knowledgeable source. In hindsight, applying patterns from a carefully selected pattern language rather than in a piecemeal way could have led to more appropriate patterns being selected for the project, which may have avoided some of the pitfalls discussed above. An example of a such a pattern language is *"Remoting Patterns"* [KiZdVo], that provides patterns for solving problems in the domain of *"Enterprise, Internet, and Realtime Distributed Object Middleware"*.

As of the current date several major project delivery milestones have been declared, and the project is now approaching it's final delivery milestone which supports the approach taken.

Acknowledgments

Thanks to Roman Pichler for suggesting that I write this paper in the first place, for providing reviews of the first two drafts, and for ongoing support and friendship. Thanks to my fellow team members on the project described for providing valuable input to the paper as well as an enjoyable working experience. Thanks to Kevlin Henney for suggesting many of the patterns talked about, for being a general font of knowledge for the project team, for providing comments and support during the writing of this paper.

Thanks in particular to Michael Kircher who shepherded this paper and provided numerous thought provoking and insightful comments both before and after the conference. Finally thank you to the VikingPLoP 2006 workshop participants for their valuable feedback.

References

[BeAn04] Kent, Beck and Andres, Cynthia, *"Extreme Programming Explained. Embrace Change"*, 2nd Edition. Addison Wesley, Boston (2004)

[BaClKa] L. Bass, P. Clements, R. Kazman, *"Software Architecture In Practice, Second Edition"*, Addison Wesley (2003)

[BusHen] F. Buschmann and K. Henney, *"Explicit Interface and Object Manager"*, EuroPLoP 2003 Proceedings, Universitätsverlag Konstanz Gmbh (2004)

[Conway68] M.E. Conway, *"How do committees invent?"*, Datamation magazine, (1968), http://www.melconway.com/research/committees.html

[Coplien95] J.O. Coplien and D.C. Schmidt (Editors), *"A Generative Development-Process Pattern Language"*, in *"Pattern Languages of Program Design"* (1995)

[Crah02] E. Crahen, *"Executor. Decoupling tasks from execution"*, VikingPLoP 2002

[GoF] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *"Design Patterns – Elements of Reusable Object-Oriented Software"*, Addison-Wesley (1995)

[Henn06] K. Henney, *"Context Encapsulation. Three Stories, a Language, and Some Sequences"* (2006),

http://www.two-sdg.demon.co.uk/curbralan/papers.html

[KiZdVo] M. Voelter, M. Kircher, U. Zdun , *"Remoting Patterns : Foundations of Enterprise, Internet and Realtime Distributed Object Middleware"*, Wiley Software Patterns Series

[MODPA] Mobile Design Patterns and Architectures research project – *"Patterns for Mobile Application Development"*,

http://www.titu.jyu.fi/modpa/Patterns/PatternList.html

[POSA1] F.Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *"Pattern-Oriented Software Architecture Volume 1 – A System of Patterns"*, John Wiley and Sons (1996)

[POSA2] D.C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *"Pattern-Oriented Software Architecture Volume 2 – Patterns for Concurrent and Distributed Objects"*, John Wiley and Sons (2000)

[POSA3] M. Kircher and P. Jain, *"Pattern-Oriented Software Architecture Volume 3 – Patterns for Resource Management"*, John Wiley and Sons (2004)

[ScBe01] Schwaber, Ken and Beedle, Mike, *"Agile Software Development with SCRUM"*, Prentice Hall, Upper Saddle River, NJ (2001)

[TTECH] Qt by Trolltech home page: http://www.trolltech.com/products/qt

[WIKSNG] Wikipedia definition of singleton – http://en.wikipedia.org/wiki/Singleton_pattern

[WIKSOA] Wikipedia definition of SOA – http://en.wikipedia.org/wiki/Service-oriented_architecture

Software Patterns

Credential Delegation: Towards Grid Security Patterns

Michael Weiss
School of Computer Science, Carleton University, Canada
weiss@scs.carleton.ca

Introduction

Security is a central concern in grid computing. A grid is a platform for sharing resources (such as computers and storage) across organizational boundaries. Thus, using a grid raises fundamental security challenges such as ensuring that only trusted organizations access our resources, data integrity and confidentiality, and delegation of credentials to grid applications. The Grid Security Infrastructure (GSI) implemented by Globus [5] and other grid toolkits [1] addresses many of these security challenges. A practical introduction to the GSI is given in the Globus Toolkit 4 tutorial [8]. Our goal is to document the patterns underlying such grid security solutions.

In this paper we describe the Credential Delegation pattern. This is one pattern from a pattern language for grid security we are writing. One of the other patterns that we will describe is Mutual Authentication, a precondition for applying the Credential Delegation pattern. Its intent is for two parties (client and server) to verify each other's identities. In our description of Credential Delegation, we follow the format for security patterns defined in [7]. Our focus is on security aspects. For a general pattern-oriented introduction to grid computing see the grid architectural pattern in [3].

Credential Delegation

In a grid parties may need to act on behalf of other parties. They should be able to authenticate themselves as acting on behalf of those parties. Therefore, issue a special type of certificate (proxy certificate) signed by the original party (grantor) that confirms that the holder of this certificate (grantee) is allowed to act on its behalf.

Example

A user at site A requests to run a simulation on a powerful server at site B. But this simulation may – on the user's behalf – need to access input data or invoke services on another server at site C. Typically, these parties (user, simulation, and providers of input data or services) are all in different security domains, and may use different local security mechanisms (Windows, Unix). Figure 1 summarizes this scenario.

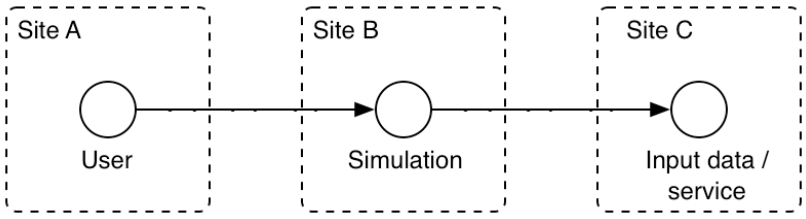


Figure 1: A user at site A requests to run a simulation at site B. This simulation needs to access input data or invoke services at site C on the user's behalf

Context

A grid in which parties need to act on behalf of other parties. You are using Mutual Authentication to establish secure communication between the parties.

Problem

When a party requests a service on behalf of another party, it should be able to authenticate itself as acting on behalf of that party. Refer to those parties as George, Fred, and Ted. Fred requests a service from George, who in turn requests services – on Fred's behalf – from Ted. One solution would be to ignore Fred's identity, but this has the obvious drawback that George may lack the credentials to request the service from Ted. Another solution would be to specify that the request is made on Fred's behalf, and for Ted to contact Fred about the validity of the request. But it is impractical to ask Fred about each service requested on its behalf. Fred may also already have signed off by the time Ted tries to contact it. Finally, George could demonstrate that it is acting on Fred's behalf, if Fred had provided its private key to George. However, this solution results in a severe security breach, as private keys must always remain secret.

The solution to this problem must balance the following forces:

- George must operate independently from Fred: Fred cannot be expected to be available, or to provide any information beyond what it initially provided.
- George, as it is acting on Fred's behalf, must be provided with a means of demonstrating to other services such as Ted that it represents Fred.
- This means should not be valid beyond the scope of the request. Otherwise, George[1] could pretend to act on Fred's behalf in the context of unrelated requests.

Solution

Issue a special type of certificate signed by the original party (grantor) that confirms that the holder of this certificate (grantee) is allowed to act on its behalf. A private/public key pair is generated specifically for such a proxy certificate, and the

[1] Or somebody else who manages to compromise the means.

authority to act on the grantor's behalf extends only to the holder of this private/public key pair, that is, the grantee. One issue to be addressed by this solution is that such a proxy certificate would allow the grantee to act unconditionally on the grantor's behalf. Above we referred to the grantor as Fred and the grantee as George.

While the grantor might trust the grantee to act on its behalf during the context of the specific request, it is unlikely to trust the grantee indefinitely. Thus, the lifetime of the proxy certificate is limited (usually to a few hours). Should some other party compromise the proxy certificate, it will now be of limited use. Some implementations of this solution (e.g. X.509) also support the placement of restrictions on the grantee by means of proxy certificate policies. For example, the proxy certificate might only allow the grantee to read certain files. This is another way of mitigating the unintended use of a proxy certificate. The use of proxy certificates results in a delegation of credentials (i.e. the grantor's identity) to the holder of the certificate (grantee).

Structure

Figure 2 shows the structure of this pattern. A `Grantor` asks a `Grantee` to perform a request. As executing the request may involve the invocation of another `Service`, the `Grantor` gives the `Grantee` permission to invoke such a `Service` on its behalf. This permission is embodied in a `Proxy Certificate`, which the `Grantee` creates and asks the `Grantor` to sign. The `Service` can verify the identity of the `Grantor` by checking the `Proxy Certificate`, that is, it does not need to contact the `Grantor`.

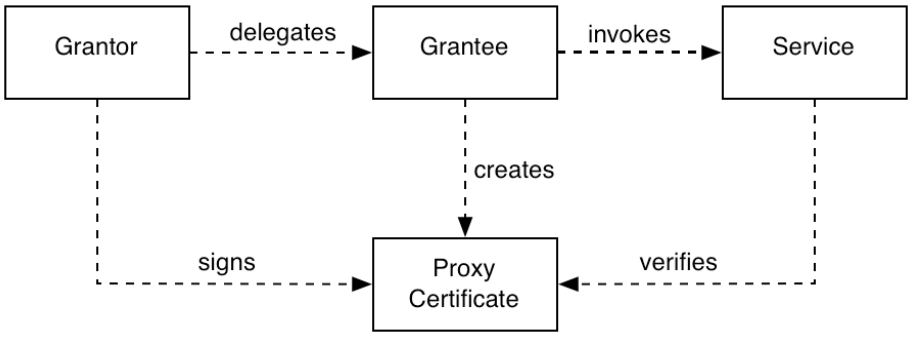


Figure 2: Structure of the Credential Delegation pattern

Dynamics

The sequence diagram in Figure 3 illustrates the process of generating and using a proxy certificate. The steps are as follows:

1. The `Grantor` and `Grantee` establish a secure channel (using SSL) to assure the integrity and confidentiality of any subsequently exchanged information.
2. The `Grantee` generates a temporary public/private key pair to be used for the `Proxy Certificate` and signs the new public key with its own private key.
3. The `Grantee` sends the signed `Proxy Certificate` request to the `Grantor`.

4. The `Grantor` signs the `Proxy Certificate` request with its private key. This results in the creation of a new certificate, the `Proxy Certificate`.
5. The `Grantor` then sends the `Proxy Certificate` to the `Grantee`.
6. The `Grantee` uses the `Proxy Certificate` to establish a secure channel with the `Service` that it needs to invoke on the `Grantor`'s behalf.

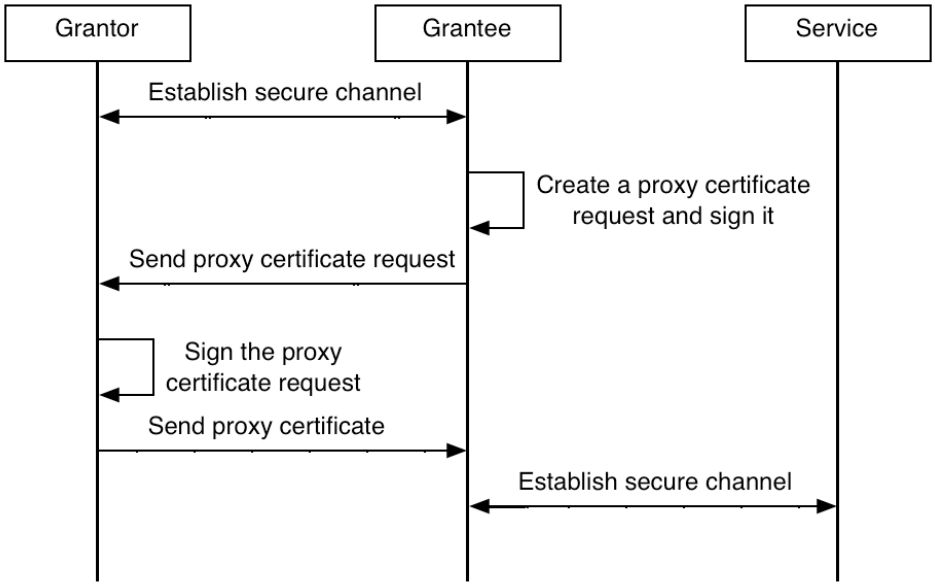


Figure 3: Generation and use of a proxy certificate

A proxy certificate is like a digital certificate, except that it is not signed by a certificate authority. Thus, for a service to establish the validity of the proxy certificate, it first verifies that the certificate was signed by the grantor, applying the grantor's public key to obtain the certificate's public key as signed by the grantee. Then, it applies the grantee's public key to extract the certificate's public key. The service obtains the grantee's public key from the grantee's certificate, which is typically sent together with the proxy certificate by the grantor, and is signed by a certificate authority.

Example Resolved

Before the simulation (grantor) at site B can access input data or invoke services at site C on behalf of the user (grantee) at site A, the simulation and user application mutually create a proxy certificate. The service request is then accompanied by the proxy certificate and the user's certificate, which allows the service at site C to verify the proxy certificate, and to establish a secure channel between simulation and service.

Figure 4 shows how the example is resolved. Using stereotypes the role of each component (grantor, grantee, service) is indicated in correspondence with the pattern.

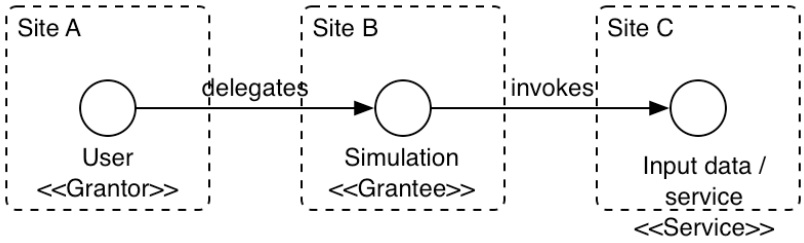


Figure 4: Example resolved (stereotypes indicate roles in the pattern)

Known Uses

The principle of credential delegation using proxy certificates was independently introduced in the Digital Distributed System Security Architecture (DDSSA) [4], and in version 5 of the Kerberos authentication system [6]. Later it became the basis for the design of the Grid Security Infrastructure (GSI) [2]. GSI is a library for public key based authentication and authorization designed for inter-domain resource sharing. It is the most widely adopted security solution for grid middleware, and prominently in the Globus Toolkit [5]. Other uses of credential delegation, which predate the development of Globus, are documented for the I-WAY system [1, chapter 4], and the Legion Grid [1, chapter 10], among others. In I-WAY, proxy user ids were used to authenticate the user to remote resources on the user's behalf. In the Legion Grid, user credentials are communicated through authentication objects, and subsequently used to access data on the user's behalf. RFC 3820 [9] provides a detailed discussion of the proxy certificate technology underlying the GSI. Finally, it should be noted that although a critical component of GSI, credential delegation is not limited to grids.

Consequences

The following benefits may be expected from applying this pattern:

- Proxy certificates allow users to delegate authority to parties that then act on their behalf, even if the user is no longer available.
- The user's security is protected due to the short lifespan of proxy certificates.[2] Compromising a proxy certificate has far less impact, in terms of damage that can be done and recovery, than if the user's own certificate had been compromised.
- Use of proxy certificates also gives us single sign-on. When interacting with multiple resources, the user only needs to be authenticated once in order to create a proxy certificate, and can use the proxy certificate for subsequent authentications.

The following liabilities may arise from applying this pattern:

[2] A participant of the writer's workshop suggested refactoring this pattern into smaller patterns. For example, this and the next benefit of using the pattern could be documented in separate patterns with potential names of Short Lifespan and Single Sign-On. As we add to our pattern language we plan to rework this pattern accordingly.

- Performance is slightly degraded, because of handshake required to create a proxy certificate, but also improved slightly on subsequent authentications.
- The infrastructure becomes more complex in order to support proxy certificates.

See Also

The pattern uses Mutual Authentication (also known as Known Partners [7]) to establish the initial trust between a grantor and the grantee to which it delegates its credentials. There are many issues not directly addressed by the pattern, such as dealing with timeouts during transactions longer than the timeout period, and delegation of user credentials through a chain of systems. They will be addressed by future patterns.

Acknowledgements

My thanks go to my tiresome shepherd Jorge Ortega Arjona. His feedback has helped me significantly in bringing the workshop version of this paper into shape. I also want to thank the participants of my VikingPLoP workshop for their thoughtful comments: Jim Coplien, Aino Vonge Corry, Yngve Espelid, Jayashree Kar, Lars-Helge Netland, Rebecca Rikner, Andreas Rüping, Kristian Elof Sørensen, and Birgit Zimmermann.

References

1. Berman, F., Fox, G. and Hey, T., Grid Computing: Making the Global Infrastructure a Reality, Wiley, 2003.
2. Butler, R., Engert, D., Foster, I., Kesselman, C., and Tuecke, S., A National-Scale Authentication Infrastructure, IEEE Computer, 33, 60-66, 2000.
3. Camargo, R., Goldschleger, A., Carneriro, M. and Kon, F., Grid: An Architectural Pattern, Conference on Pattern Languages of Programs (PLoP), 2004.
4. Gasser, M. and McDermott, E., An Architecture for Practical Delegation in a Distributed System, Symposium on Research in Security and Privacy, 20-30, IEEE, 1990.
5. Globus Alliance, GT4 (Globus Toolkit 4) Security: Key Concepts, http://www.globus.org/toolkit/docs/4.0/security, 2006.
6. Neuman, B., Proxy-Based Authorization and Accounting for Distributed Systems, International Conference on Distributed Computing Systems, 283-291, 1993.
7. Schumacher, M., Fernandez-Bugliono, E., Hybertson, D., Buschmann, F. and Som- merlad, P., Security Patterns, Wiley, 2006.
8. Sotomayor, B. and Childers, L., Globus Toolkit 4: Programming Java Service, Morgan Kaufmann, 2006.
9. Tuecke, S., Welch, V. et al, RFC 3820 – Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, 2004, http://www.faqs.org/rfcs/rfc3820.html.

Security Pattern for Input Validation

Lars-Helge Netland, Yngve Espelid, Khalid Azim Mughal
Department of Informatics, University of Bergen
{larshn, yngvee, khalid}@ii.uib.no

Abstract

This paper discusses a security pattern for input validation in web applications. A template description facilitates understanding of the important concepts, and allows developers with a security background to quickly adapt the pattern in their own context.

1 Introduction

E-commerce has become a major factor in the marketplace. In Norway, the percentage of total turnover from E-commerce grew from 2.2% in 2002 to 3.9% in 2005 [1]. This rapid growth has fueled a need for new software for selling goods and services over the Internet. In creating such systems, security aspects have been set aside in favor of time-to-market concerns and new application functionality. As a consequence, massive quantities of sensitive information lie inadequately protected on the Web.

Enter the villain. The Internet is a candy store for computer literate criminals. Snacks are readily available in form of credit card numbers, classified corporate data, and sensitive medical information. A long-time favorite target among hackers is the web-based shopping cart. Many developers use hidden fields to store and camouflage data from users, and fail to realize that the information can be easily manipulated. Through such tampering attacks, attackers can dictate their own prices in the web store, and hence purchase goods at extreme discounts. This vulnerability was reported in public no later than February 2000 [2], but software developers continue to release applications that are wide open to such attacks [3].

Software security is not a new research area. Saltzer and Schroeder's pioneering work "The protection of information in computer systems" [4], describes basic principles that remain true even today. More recently, much effort has gone into describing how attackers break software. In terms of building secure software, best practice guidelines have dominated the security literature. Patterns have become a popular way of sharing structured knowledge and experience from successful projects in traditional software engineering. A similar approach looks promising for distributing software security knowledge. The first paper in this direction was written by Yoder and Barcalow [5] in 1997.

Input handling is the process of checking data supplied by others against a set of predefined rules. The Open Web Application Security Project (OWASP) [6] has defined 3 categories of input handling:

Integrity checks, ensure that data has not been tampered with.

Validation, a set of rules that make sure that data is of a certain type, has correct syntax, is within specified length boundaries, or contains only permitted characters.

Business rules, checks that enforce back-end business logic, such as disallowing due dates already passed in an online banking application when paying bills.

The Input Validator pattern described in this paper addresses *validation* and *business rules*. It should be noted that specialized attacks such as SQL injection and cross-site scripting can be dealt with more effectively through prepared statements and HTML encoding, respectively [7]. It is possible to design a filter to look for metacharacter attacks, but identifying valid input (positive validation) is considered better than looking for data that can do you harm. Choosing the latter strategy is more costly in terms of maintenance, where you have to be constantly up-to-date with new threats, while the former option can provide protection against some of the attacks to come.

Unvalidated input is number one on OWASP's top ten list over web application vulnerabilities [8]. The rest of the paper presents a security pattern for input validation. We follow the template used in [9, p. 9], which is the de facto standard for writing security patterns.

2 Input Validator

A number of attacks on web applications target the application protocol through which vendors conduct business. Firewall technology and a Protection Reverse Proxy [9, p. 457] can enforce access control rules, but usually do not stop exploits embedded in message content. The back-end server(s) needs an Input Validator to validate data sent from the client.

Example

An Internet banking provider has recently been plagued by a series of hacker attacks. It started a few weeks back when about a dozen of the bank's clients filed reports demanding to know why their accounts had been cleared out. In the investigation that followed another problem was discovered: a fair quantity of bills paid by customers contained negative amounts. Fortunately, the bank invested heavily in a highly praised logging system a few months back, so sorting out the details should not be a problem. Section 2 describes how the bank improved their system's security.

Context

The input validator pattern applies to services offered through a client-server model communicating over HTTP. The goal is to validate input from the client. Input validation does not make network security solutions obsolete. It is a supplementary defense mechanism that helps to secure the environment in which the system runs. Any so-called client-side validation checks are worthless from a security standpoint. They serve the following three purposes: a) Enhance user experience through instant feedback on non-conforming input, b) conserve network bandwidth by reducing requests to the server, and c) save processing resources on the server. The term client-side validation is misleading, as none of the benefits mentioned above are related to security.

Problem

Many software systems fail to inspect and filter untrustworthy input. Through cleverly crafted input hidden in message content, attackers can alter the program logic and possibly exploit implementation weaknesses for economic gain. The pertinent question to ask is "How can one defend client-server applications against attacks hidden in the message content?"

Forces

The input validation solution must resolve the following forces:

- Extensive examination of input means more computational overhead. Developers should strive to make input validation as non-intrusive as possible. This is especially so in data-intensive systems, where delays that are acceptable in smaller applications, get compounded and render the system useless.
- The constant tug of war between software developers and malicious hackers has repeatedly shown how a dedicated attacker community discovers new ways of exploiting software. Systems should be designed with this ongoing battle in mind and incorporate flexible solutions that can be easily extended, if necessary, to counter future threats.
- Usability is a key factor for security constructs to prevail. Systems that are complex and difficult to operate cause maintainers and users to do mistakes that jeopardize security.

Solution

Perform syntactical and semantical validation of all input that is to be processed on the server. These 2 categories of validation are termed "validation" and "business rules" by OWASP. This process entails constructing appropriate validation rules for each source of input.

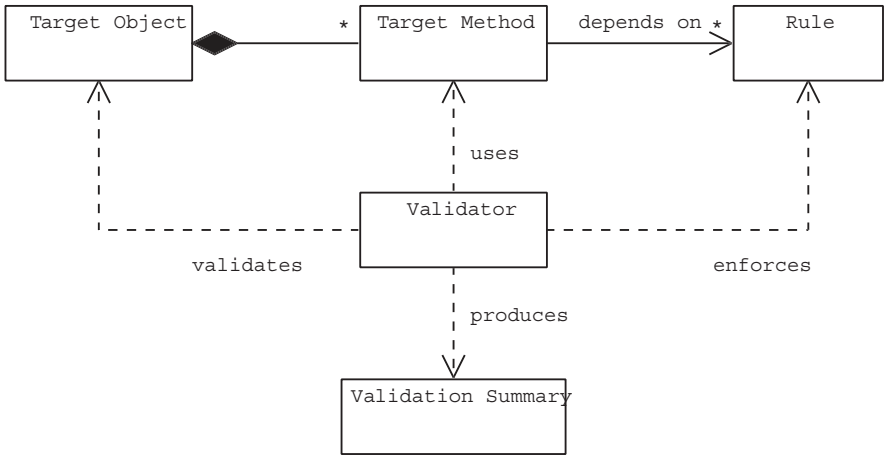


Figure 1: Content validation entities

Structure

Fig. 1 shows the class diagram for this pattern. The primary entities are

`Target Object`, which is the target for the validation process, comprising properties subject to inspection.

`Target Method`, which exposes an object property.

`Rule`, which defines constraints for the exposed object properties.

`Validator`, which controls the validation process, validating the exposed properties of an object against sets of rules and reporting any rule violations.

`Validation Summary`, which provides a summary of all rule violations.

Dynamics

The workings of an Input Validator can be explained in terms of an airport security checkpoint. Fig. 2 shows a typical setup, where travelers must pass a metal detector frame, and carry-on items are inspected in an X-ray machine. In this context, the primary entities can be interpreted as follows: The `Validator` is the airport security personnel and equipment; `Target Objects` are represented by airline passengers; `Target Methods` correspond to individual inspection targets, such as briefcases, cameras, or laptops; `Rules` express which items should be disallowed, typical examples include knives, guns, and can openers; `Validation Summaries` would be incident reports produced by the airport security personnel in the event of rule violations.

Consider the sequence of steps executed when a passenger shows up at the security checkpoint. First, the commuter must show a passport and valid travel documents to prove that he or her has purchased a service from one of the airlines

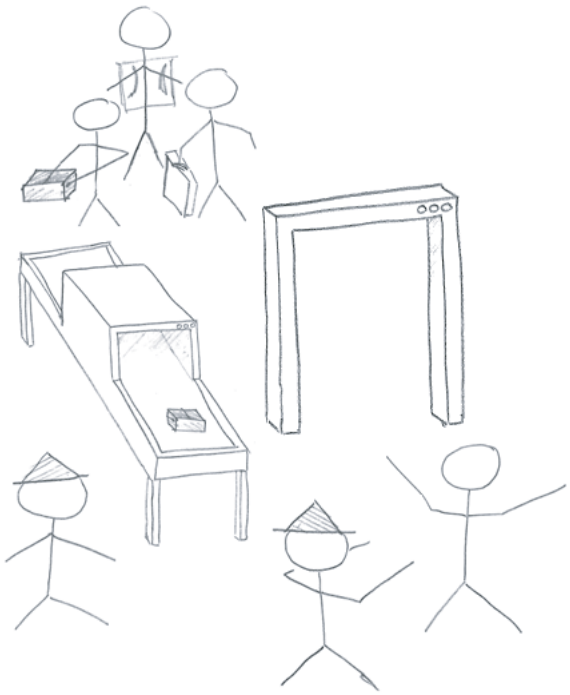


Figure 2: Passenger inspection at an airport security checkpoint

inside the security boundary. This is a precondition for input validation, and can be thought of as a firewall mechanism. Next, the security officials examine the traveler for illegal items, in accordance with a predefined list of disallowed objects. The passenger walks through a metal detector, while clothes, luggage, and other accessories are inspected with X-rays. If the airport security personnel finds it necessary, additional measures such as bomb-sniffing dogs and full body searches may be used. Travelers that do not carry prohibited items can continue their journey, while perpetrators are guided away from the security checkpoint and face further inquiries. Personnel at the checkpoint report offenses to other institutions, such as the police. Likewise, the INPUT VALIDATOR generates a `Validation Summary`, which in turn is handled by the application.

The process of explicitly specifying illegal items is more commonly referred to as *blacklisting*. This is a useful technique when you have a complete overview of the potential enemies of your system.

Implementation

Fig. 3 shows an example implementation of the INPUT VALIDATOR using firewalls [9, p. 403] and an INTEGRATION REVERSE PROXY [9, p. 465]. To implement this pattern, the following tasks should be performed

1. Identify all potentially vulnerable subsystems. Developers must single out components where message content will be processed. In Fig. 3, the stock application and the account manager can be exploited.

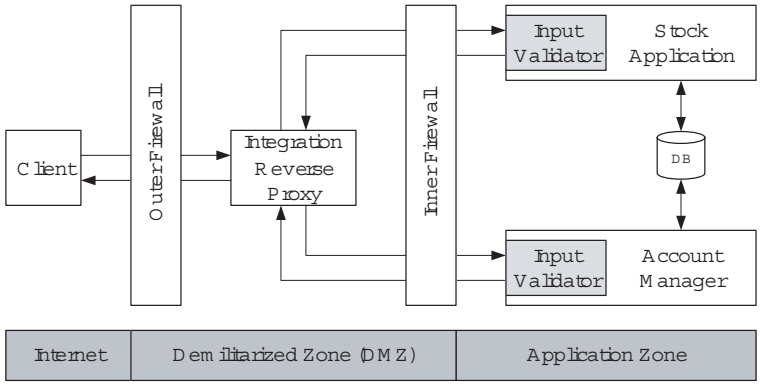


Figure 3: Example content validator architecture

2. Determine the entire set of client input sources and define what is to be considered valid input. We recommend using a whitelist approach [7, p. 71] to specify the format of valid input, also known as *positive validation*. All data not conforming to the format should be discarded.

3. Configure target objects and methods for the validators using the set of client input sources. Construct rules from the valid input definitions from the previous step, and map these rules to the corresponding target methods.

4. Deploy the validators. As shown in Fig. 3, validators should be integrated into every potentially vulnerable web applic ation.

The input validator pattern should be combined with an accounting mechanism which can capture and store information about rule violations, and enable a reviewer to identify the involved participants and reconstruct the events that led to the violations. Security Accounting Requirements [9, p. 360] can help in selecting an appropriate accounting mechanism.

With a focus on functionality and user experience, many web application developers try to scrub content on behalf of the client. If violations are reported during validation of a newly received request, programming logic tries to alter the input with the goal of making it valid. Often, this scrubbing logic is complex and may incur significant processing overhead. A better alternative is to use the logs to evaluate the validation rules, and decide if the rules should be altered to cover a larger set of input.

Example Resolved

The Internet bank mentioned in Section 2 was able to determine the cause of the anomalies by inspecting the server logs. The emptying of client accounts was caused by a brute-force attack launched against the bank's authentication mechanism. During the analysis it also became clear that the system lacked validation of business rules.

To rectify the situation the bank integrated an input validator into their web application. The following steps were taken:

1. The authentication mechanism was improved by increasing the password from 4 digits to 6 characters, with the minumum requirements of one lower case letter, one upper case letter, and one digit. All customers were forced to change their passwords using a web form presented at their next log in. The following regular expression rule was created to enforce that new passwords adhered to the new policy:

   ```
   ^(?=.*?[a-z])(?=.*?[A-Z])(?=.*?\d)\S{6}$
   ```

2. A number of business rules were developed to enforce business logic when paying bills. For instance, a rule only allowing numbers greater than zero was specified for the amount field. To prevent customers from entering due dates in the past, the following rules were added:

   (a) A regular expression rule defining `dd.mm.yyyy` as the valid date format:

   ```
   ^\d{2}\.\d{2}\.\d{4}$
   ```

   (b) A rule for checking that the due date is the current date or in the future

Known Uses

A number of worked solutions for input validation exist

Commons Validator [10] is an open-source validation component which originated from the Apache Struts framework. Target objects are JavaBeans. Rules are named *validator actions* and are individual static boolean methods in Java objects. The component enables developers to configure target methods and rules in configuration files.

Stinger [11] is an HTTP Request Validation Engine used in a J2EE environment. The HTTP request is the target object. The engine supports regular expression rules and rules for missing or extra parts in HTTP requests. Developers can specify the rules using the Security Validation Description Language, which is tailored to map the target methods in HTTP requests.

.NET Validation Server Controls [12] are part of Web Forms in the .NET framework, and enable developers to perform input validation on client input in web applications. The target objects are the input server controls, and the framework provides predefined rules such as required field, ranges, and regular expressions. The validation controls are specified as part of the presentation logic.

Consequences

The following benefits can be expected from applying this pattern:

- Input validation can prevent attacks embedded in message content.

- Whitelisting wards off new unknown attacks that fall outside the format dictated by the validators.
- The mechanism enables accounting of attacks by capturing rule violations. The information can later be used to reconstruct attempted security breakins.
- The input validator provides a centralized and manageable mechanism for input validation.
- The same validation rule can be applied to multiple target methods. Input sources that share the same characteristics should reuse the same validation logic.

The following potential liabilities can arise from applying this pattern:

- A client request will carry additional computing overhead, causing delayed server response. The addition of validation logic also causes the server to reach its processing limits sooner.
- The added components introduce new points of failure. Potential bugs can make the application unavailable.
- The new layer of security increases system complexity, which in turn can entail a higher maintenance cost.

See Also

Client Input Filters [13] discusses the input validation problem in a web context. It points out the inadequacy of client-side checks, and argues that the same checks must be carried out on the server-side as well. The pattern focuses on best practices and issues in implementing web applications.

Intercepting Validator [14] is a J2EE pattern for validating client input. It is not tied to business logic, and attempts to filter out known attacks early in the request-handling process. In contrast with the Input Validator, the authors recommend using the Intercepting Validator to also protect against injection attacks.

Useful patterns that can be beneficial in conjunction with the Input Validator pattern include accounting, firewall, and patterns for secure Internet applications [9].

Acknowledgements

Our sincere thanks to Andreas Rüping, who did a great job in shepherding our paper, providing invaluable feedback. Also, we would like to extend our deepest gratitude to the fellow members of our writers workshop group at VikingPLoP 2006: Kristian Elof Sørensen, Aino Vonge Corry, Birgit Zimmerman, Andreas Rüping, Michael Weiss, James O. Coplien, Jayashree Kar, and Rebecca Rikner.

References

[1] Eurostat. Retrieved May 2006 from `http://epp.eurostat.cec.eu.int/portal/page?_pageid=1996,39140985&_dad=portal&_schema=PORTAL&product=_STRIND&root=theme0/strind/innore/ir080&zone=detail`.

[2] ISS E-Security Alert: Form Tampering Vulnerabilities. Retrieved June 2006 from `http://www.cert-rs.tche.br/listas/infoseg/msg00033.html` .

[3] Open Source Vulnerability Database. Retrieved June 2006 from `http://www.osvdb.org/displayvuln.php?osvdb_id=18449` .

[4] J.H. Saltzer and M.D. Schroeder, "The protection of information in computer systems," in *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[5] J. Yoder and J. Barcalow, "Architectural Patterns for Enabling Application Security," in *Proceedings of Pattern Languages of Programs (PLoP)* , 1997.

[6] Data Validation - OWASP. Retrieved June 2006 from `http://www.owasp.org/index.php/Data_Validation` .

[7] S.H. Huseby, *Innocent Code—A Security Wake-Up Call for Web Programmers.* John Wiley & Sons, first edition 2004.

[8] OWASP Top Ten. Retrieved May 2006 from `http://www.owasp.org/documentation/topten.html` .

[9] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns—Integrating Security and Systems Engineering*. John Wiley & Sons, first edition 2006.

[10] Commons Validator. Retrieved May 2006 from `http://jakarta.apache.org/commons/validator/` .

[11] Aspect Security, Stinger HTTP Request Validation Engine. Retrieved April 2006 from `http://www.aspectsecurity.com/stinger/` .

[12] Validation Server Controls. Retrieved May 2006 from `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconASPNETSyntaxForValidationControls.asp`.

[13] Client Input Filters. Retrieved May 2006 from `http://www.scrypt.net/~celer/securitypatterns/`.

[14] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns—Best Practices and Strategies for J2EE, Web Services, and Identity Management.* Prentice Hall, first edition 2005.

Software Architectures for Web Content Management

Patterns for Deployment and Infrastructure

Andreas Rüping
Sodenkamp 21 A, 22337 Hamburg, Germany
andreas.rueping@rueping.info
www.rueping.info

Introduction

Web content management comprises technologies and processes around the creation and maintenance of digital content and its delivery to the web. Content management systems can help you with that, but if you develop a large-scale site it's highly likely that it will also require a good amount of individual software development. This is true especially if a good amount of flexibility is needed for an individual information model, the design of individual templates, the integration of third party-products such as search engines, or for personalisation. The site will probably also face non-functional requirements such as maintainability, scalability or performance, so you'll quickly find yourself in the middle of a discussion of software architecture issues.

This paper is an excerpt from a larger collection of patterns that take a look at the software architecture side of web content management. The patterns don't describe how to build a CMS tool, but address the development of your own software on top of a CMS of your choice. This paper contains the Chapter V of the complete collection and deals with deployment and infrastructure. Typical discussions of this paper address web application definitions, web server configurations, environments (development, live and otherwise), development processes, and so on. Throughout this paper I'll use a running example to explain how the patterns work. The example is a web portal for technology transfer. The details should become clear as we go into the actual patterns.

The complete collection of patterns is very much a work in progress. To give you an idea of the overall collection in its current form I have added thumbnails of the other patterns in the Appendix. Moreover, there are two EuroPLoP papers that cover chapters from the overall collection that also give some insight in tis ongoing effort (Rüping 2005, Rüping 2006).

It's likely that the collection will evolve further and I'd like to invite everybody interested in sharing their views to get in touch with me. Feel free to contact me at "andreas.rueping@rueping.info", whether it is with agreement, good ideas, concrete suggestions for improvement or experience reports.

Acknowledgements

Thanks go out to Michael Weiss who, as the VikingPLoP 2006 shepherd for this paper, came up with much useful feedback that has helped me improve the paper for the conference, and that will also be useful when one day the full manuscript undergoes a major revision.

Thanks also to Uwe Zdun for his comments on the first draft of the full manuscript (including this chapter) that he shepherded for EuroPLoP 2004.

Last, but certainly not least, I'd to thank the VikingPLoP 2006 workshop where this paper was discussed. It's always amazing how much good feedback you receive at such a workshop — thanks to everybody who shared there ideas and suggestions.

V. Deployment and Infrastructure

5.1 One Web Application

Context You have implemented the custom server-side components for your web site. This probably includes a system of INTERACTING TEMPLATES (2.3) for the actual HTML generation as well as components that encapsulate additional functionality, for instance for navigation, searching or personalisation. You have to deploy all these components when you launch the site since the site relies on these components in their entirety.

Problem **How many web applications should you define for the components that you have developed for rendering content?**

Example Though not an overly complex web site, the Technology Transfer Portal consists of several components nonetheless. There are going to be quite a few templates, as well as components for various purposes.

What these components are depends on the underlying technology. If we follow a Java-based approach then there are going to be JSPs and Java classes. In an XSLT-based approach there are going to be XML files, XSLT transformations and Java classes as well. In a PHP-based approach we'll need various PHP classes. To some extent, these approaches can even be combined.

In addition, there could be classes from web frameworks such as Struts or Spring, or third-party components like search and personalisation engines.

All in all, this amounts to quite a number of components. When we deploy these components to the web, what kind of packaging is recommended?

Forces In order to launch a web site, all its components have to be deployed to a machine where they are available for the web server. The web server has to answer all http requests it receives, and therefore it has to find all necessary resources, whether they are classes, templates or configuration files, and so on.

Depending on the technology you use you may have the option to define one or more web applications. A web application consists of resources that together provide a specific service to the users. Different web applications are relatively independent of each other, while the resources within the same web application collaborate closely. The concept of web applications is perfectly common in the Java world. It's supported by virtually every web server and application server on the market. Other technologies such as PHP don't yet offer support for separate web applications, though frameworks such as Zend are heading in this direction.

Different web applications bear the advantage that they can be deployed separately. You can launch the new version of one web application, while all other web applications remain untouched. This allows you to partition your web site or portal into fairly independent parts and avoids a big-bang style deployment which is extremely error-prone. You can use separate configuration files for your web applications, specify separate log levels and use separate name spaces.

Establishing separate web applications, however, has its drawbacks too. Because separate web applications are largely independent of each other, collaboration between them is possible but limited.

First, calling a component from another web application is possible, but more complicated and usually less efficient than calling another component inside the same web application. Second, application servers maintain different sessions for different web applications — exchanging session information across web applications is possible but relatively awkward.

Many of your custom components, however, rely on each other heavily. Several of them call each other, some even have to share the same session information. Scattering these components over several web applications would introduce significant dependecies between these applications and therefore doesn't seem like a good idea. It's for cases like this that Paul Dyson and Andy Longshaw, in their book on *Architecting Enterprising Solutions*, recommend to "group all core system functionality in a single application" (Dyson Longshaw 2004).

Solution

Define one web application for all server-side components that are involved in the page generation and delivery for your site. This includes both templates and any service components that you may have developed, as well as components provided by the content management system. It is fine to single out components in separate web applications if their functionality is only remotely connected to page generation.

In detail, the central web application includes the following components:

- the templates for all document types, essentially one TEMPLATE PER VIEW (2.4)
- the DOCUMENT WRAPPER (2.1) component for each document type
- the NAVIGATION MANAGER (2.2)
- the components specific to the search functionality available for your site, including the SEARCH REQUEST CONTROLLER (3.3), the SEARCH ENGINE ADAPTER (3.4) and the SEARCH MANAGER (3.5)
- all components specific to the personalisation available for your site, such as a ROLE-BASED CONTENT FILTER (4.1) and a USER MANAGER (4.2)
- the components from your content management system that collaborate with your custom components
- all necessary configuration files

Grouping all these resources into one web application removes the barriers that could hinder their close collaboration.

Candidates for separate web applications include modules whose functionality can be regarded more or less independently of any web page generation. Examples include the search engine (usually a distinct CMS component or a third-party product), a payment component, or some external single-sign-on solution that you may use.

Content validation is quite a different matter. You don't apply TYPE-SPECIFIC VALIDATORS (1.6) in the process of HTML generation but instead integrate them into the content editors' workflow. It's therefore clear that validators don't belong to the core web application.

Example Resolved

We introduce one central web application for the Technology Transfer Portal. The one component that is not going to be part of that web application is the search engine. The search engine is sufficiently independent and it's only connected via a SEARCH ENGINE ADAPTER (3.4), so it's fine to regard it as an application of its own.

How the central web application is organised depends on the technology we use. It's clear that individual content management systems have their specific requirements concerning deployment structures. Assuming the Java world, Figure 1 shows, as an example, how this web application can be represented in the file system structure if we use Tomcat as our application server. You can see that there's one web application named "technology-transfer" with subdirectories for templates, Java classes, libraries and so on.

Benefits

+ The amount of cross-application communication is clearly reduced. First, this makes collaboration between components more straightforward and more maintainable. Second, there is a positive impact on the system performance due to a smaller overhead caused by cross-application calls.

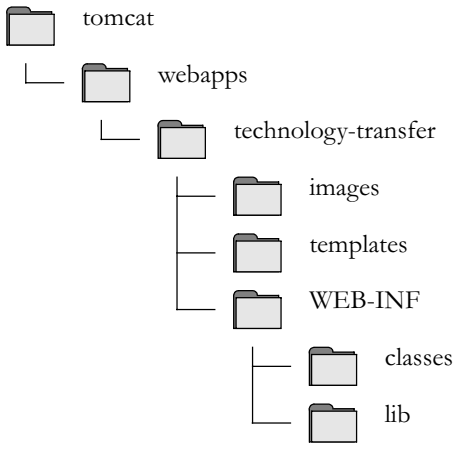


Figure 1 Deployment structure in a Java world

Liabilities – Having just one web application can make a hot deploy difficult. With the exception of things like templates and scripting elements, you usually can't exchange individual parts of a web application at run time. If, for instance, you want to replace certain Java classes involved in page generation or delivery, you'll have to re-start the entire system. Introduce DEDICATED DEVELOPMENT AND PRODUCTION ENVIRONMENTS (5.4) to alleviate this problem, and establish COORDINATED RELEASE CYCLES (5.5) to manage dependencies between software deployment and content publishing.

5.2 Legal Request Policy

Context You have implemented a system of INTERACTING TEMPLATES (2.3) that generate the HTML pages that make up your site. Now you deploy these templates to the web server environment, along with all other components.

Problem **How can you prevent users from retrieving incomplete page fragments, or else from invoking unauthorised requests?**

Example The Technology Transfer Portal offers a set of pages linked to each other through the site navigation. Obviously we want users to read these pages, enjoy them, retrieve the information they seek.

We don't want users to get to see isolated page elements though. Individual page elements, such as a menu, a login form or a text heading should always appear in the context of an overall page, but shouldn't appear isolated.

In particular, it must be impossible for users to receive isolated page elements that would normally appear in the context of a page that requires authentication. Examples include text blocks from articles in the protected area, as well as the full navigation tree with links to the protected area which should only be available to authenticated users.

Forces After the web server has received an http request for a specific web page, the content management system invokes the template that is responsible for rendering that page. It depends on your content management system how the necessary mapping from pages onto templates is maintained. Suffice it to say that such a mapping is implemented somewhere, and that it's possible to invoke the right page template in response to an http request.

If you have implemented a system of INTERACTING TEMPLATES (2.3), then there is probably an outermost template that calls other templates that in turn render the individual page elements. What if, intentionally or by accident, a user makes a request for such an 'inner' template directly?

An inner template probably won't deliver a complete HTML page since the tags that declare a page as well as its header and body will be missing. This might be fine, especially if that template yields a page fragment that is meant to be embedded into other sites as well. However, the template might simply generate some HTML fragment that cannot (or shouldn't) be used separately. This is clearly undesirable.

Worse yet, calling specific templates can represent a violation of access control. It's common enough that the responsibility of invoking user authentication lies with the outermost page template — the 'inner' templates that generate page elements simply assume that authentication has taken place. This is, for instance, the idea behind ROLE-SPECIFIC TEMPLATES (4.4) — a dispatcher template checks the user's role (which includes authentication) and calls specific templates depending on this role. For performance reasons, the specific templates must not trigger off any kind of authentication.

However, if users can request any ROLE-SPECIFIC TEMPLATES (4.4) directly, they may be able to bypass authentication and to retrieve information that should only be available to authenticated users.

There are other ways for users to send requests that might cause trouble. For instance users might add parameters to a URL, which might have an effect on the content management server. If, for instance, the content management system is Wiki-based, adding a parameter such as 'mode=edit' might give the user write access. This is admittedly an extreme example, and merely adding a parameter will not typically suffice to bypass all authentication, but it could still lead to unwanted effects.

Solution **Devise a policy that states which kinds of requests are legal and which aren't. Usually requests for full pages are considered fine, as are requests for page elements that may be integrated into web pages elsewhere. On the other hand, the execution of arbitrary commands or scripts typically leads to unwanted effects, and should therefore be considered illegal. Configure your environment in such a way that it accepts only legal requests.**

There are different ways to make such a configuration, depending on the underlying technology of your system. The simplest techniques are targeted directly at http requests before they even reach the content management system:

- You can configure your web server to accept only certain kinds of requests depending on a URL pattern.
- You can configure your applications server (or whatever container you use) to accept only certain kinds of requests depending on either a URL pattern or the path to the script that is to be executed.

Either way, you can ensure that direct calls for certain templates will be denied. Whether they are JSPs, XSLT transformations, PHP scripts or something else, templates can now only be called from within the container — usually by content management system components or by other templates.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>JSP pages</web-resources-name>
        <url-pattern>/templates/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
    </auth-constraint>
</security-constraint>
```

Figure 2 Security constraint for a servlet engine

In addition, you can easily configure your system to ignore possible parameters added to a URL, or at least interpret them with care, so as not to invoke actions that shouldn't be invoked.

There might be other techniques as well — check your content management system and your web and application server infrastructure how the necessary configuration can be made.

Example Resolved

There are various ways how we can rule out direct calls of any of our templates. For instance, if we use Apache as our web server we can make the necessary configuration in a file named ".htaccess" where we can use regular expressions to specify certain URL patterns (such as all requests for a JSP or a PHP script).

If we're using Java technology, we can also use the servlet engine's configuration to ensure that no templates are called directly. One option is to place all templates (usually JSPs) in a subdirectory under "WEB-INF", where they cannot be accessed directly, but can be called from other components.[1] A second option is to define a security constraint in the deployment descriptor named "web.xml". Figure 2 shows an example that covers all templates (specified by their path) and assigns no roles as being allowed, which the servlet container interprets to mean that no templates must be accessed directly.

As a consequence, unwanted requests such as ".../technology-transfer/templates/renderNavigationTreeLoggedIn.jsp" will not be processed but lead to an error message instead.

Benefits

+ Users won't receive any ill-formatted page fragments made of incomplete or invalid HTML.
+ Users cannot bypass authentication mechanisms and directly invoke templates or other components that should only be invoked after successful authentication.

1. Apparently not all J2EE servlet containers have implemented this feature.

Liabilities

– Configuring your environment can be a tricky business, especially if several machines are involved. Chances are that you'll introduce DEDICATED DEVELOPMENT AND PRODUCTION ENVIRONMENTS (5.4) to decouple software maintenance from content editing — it's crucial in this case that these environments are identical (down to the configuration level) so that the access configuration that you have tested in the production environment also works well for the live system.

5.3 Dedicated Production and Delivery Environments

Context

You're planning the system infrastructure for maintaining and operating your site.

Problem

How can you make your system scalable and at the same time protect it from attacks?

Example

It's unclear how many users the Technology Transfer Portal is going to have. And although the portal doesn't face any special performance requirements, it should be reasonably efficient whatever the number of users. We may have to scale it to a higher performance should the number of users increase with the time.

As for security requirements, it is clear that we need to protect our site from potential attackers. Obviously no user should be given direct access to the content repository. But even if an attacker gained access to the machine that hosts the web server (to which all users must have some kind of access), the attacker still shouldn't be able to enter the content repository or any other software components.

Forces

While content is created, edited and maintained, it's made available only to a limited group of people — the content editors who are responsible for its production. In many cases a workflow component manages the processes around content production. The classical 4-eye workflow is quite typical — it assumes that a content editor creates and edits an article and that an editor-in-chief performs a final quality assurance before the article is published. Of course there can be other kinds of workflow too, for instance a more complex workflow that involves more people or some kind of short-cut that involves only one person.

Whatever workflows you'll need for your site, and whatever mechanisms the content management system of your choice offers to implement these workflows, you must set up an environment in which these workflows can be integrated and that scales well to the necessary number of content editors.

Once content is published, however, the demands on the environment change. The published version of the content is no longer subject to maintenance, so no content editors need to have access to it. Instead the content will be accessed by templates and other components in response to requests received by the web server from the site visitors. Published content must therefore be made available in an environment that is sufficiently powerful to process all incoming http requests.

This leads to both performance and scalability requirements. On the one hand, the web must be able to deliver pages fast enough (in terms of efficiency criteria that you'll need to define). On the other hand, the environment must scale to a possibly increasing number of users.

Furthermore, the moment that content is published is also the moment that it is exposed to a mass of anonymous users in the Internet. To make it possible for the world to visit your site and view your pages, you must allow user requests to trigger off processes on your machines. But there has to be a limit to that. Everybody knows that requests from the outside can represent a security threat, so it's essential to take the necessary actions to protect yourself from attackers who try to break into your system via the web server that is meant to make published content available to the users.

Solution

Establish separate environments for content production and maintenance on the one hand and for content delivery on the other. Equip the production environment with the infrastructure necessary to host the workflow processes for content maintenance. Set up a demilitarised zone for the delivery environment and equip it with the infrastructure necessary to process requests from the expected number of users. Except for hardware and security configurations, both environments should be identical.

You will have to set up the necessary hardware for two separate environments and to install the necessary software components for each. This involves the following tasks:

- Design the production environment to meet the needs of the content editors. The production environment requires all CMS components, a database, a web server, an application server (or whatever container is required for executing templates and other components) and all your custom software. Complete page generation must be possible for preview and testing reasons. In addition, the production environment hosts the server-side WORKFLOW-BASED VALIDATION (1.7). If there are many content editors, you may have to use several machines instead of just one.
- Design the delivery environment according to the expected number of users and the required response time. Larger sites usually require several server machines and the necessary load-balancing mechanisms (Dyson Longshaw 2004). Each server must be supplied with a complete CMS installation, a web server, an application server and, again, all custom software components necessary for page generation. This includes all templates and service

components directly or indirectly involved in page generation — essentially all components included in the ONE WEB APPLICATION (5.1) concerned with page generation as well as stand-alone components such as a search engine.

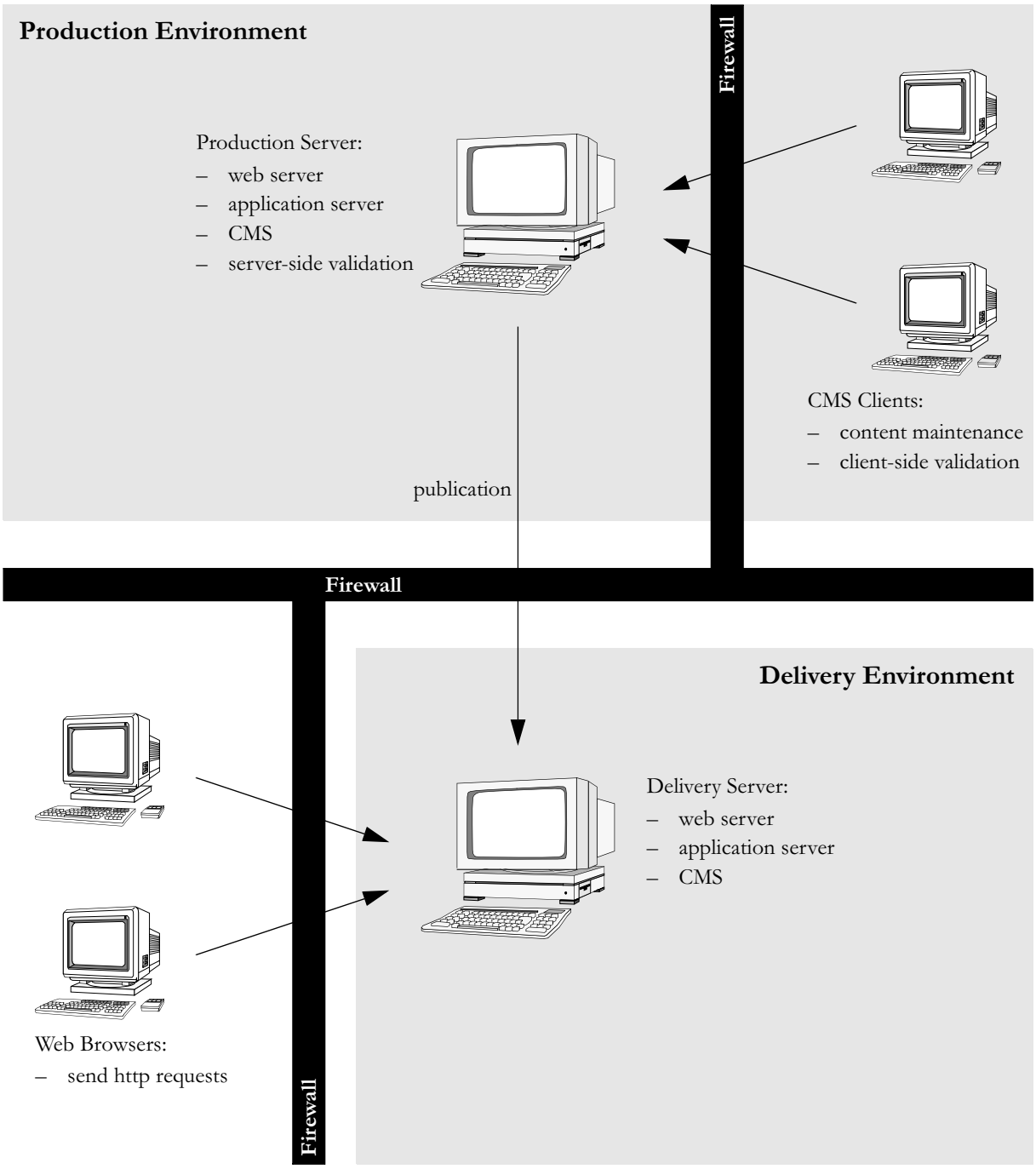


Figure 3 Dedicated production and delivery environments

- Install the necessary firewall software to make the delivery environment part of a demilitarised zone (DMZ). Limit network traffic so that only the web servers within the DMZ can be reached from outside your organisation (Dyson Longshaw 2004). Also ensure that the web server processes can only invoke content management functions in the delivery environment, but cannot gain unwanted access to machines in the production environment.
- Integrate a content transfer into the server-side workflow processes so that content, upon publication in the production environment, is made available in the delivery environment.

Figure 3 illustrates the production and delivery environments. Both environments are represented by just one machine, but as mentioned before, you can scale either environment to consist of several machines.

Details concerning the necessary hardware equipment rely very much on the specific requirements of your individual project. How many servers you need really is a question of infrastructure architecture and as such is beyond the scope of web content management. You can find valuable advice in the book by Paul Dyson and Andy Longshaw on *Architecting Enterprise Solutions* (Dyson Longshaw 2004).

Example Resolved

We set up distinct production and delivery environments for the Technology Transfer Portal. For the time being, we use one machine in either environment, though we retain the option to install additional servers in the delivery environment should the number of users exceed our current expectations.

Both environments are protected by firewalls that limit network traffic as indicated in Figure 3, with one exception. When introducing personalisation into our site we developed a PROFILE MAINTENANCE FORM (4.6) that allows registered and authenticated users to update their profiles. Profile updates submitted by this form have to be stored in the database in the production environment. We therefore have to open our firewall between the delivery and the production environment for this very specific functionality.

Benefits

+ Content editors can create and maintain content in an environment suited for their needs. This environment offers the complete functionality for page generation, which gives content editors the possibility to obtain previews for all pages. Furthermore, the production environment can integrate all necessary WORKFLOW-BASED VALIDATION (1.7).
+ Because the production and delivery environment are near identical, previews in the production environment are highly accurate. One http request should lead to mostly the same response, whether it is made in the production or in the delivery environment.
+ The delivery environment scales well to an increased number of users. You can add more server machines if necessary. Load-balancing software can distribute requests among all servers you have installed.

+ The introduction of demilitarised zone clearly contributes to system security. It is much more difficult now for attackers to damage content or software in the production environment, since firewalls actively shield this environment from the outside world.

Liabilities

– Separate environments for production and delivery require more machines and more infrastructure and therefore lead to increased costs for your site. These costs are justified by the scalability and security that you gain from this pattern, but there's no way to deny that these benefits don't come for free.

– In most cases visiting a web site means read access to the content and other data only. Sometimes, however, sites allow users to submit information that is then stored persistently in a database. Examples include email addresses submitted for a newsletter subscription, user profiles maintained by users themselves, or even arbitrary files in a site that offers an upload mechanism. In such a case you must configure the firewalls to allow for a certain degree of write access in the production environment. Although opening the firewalls somewhat impairs your security concept, it is necessary if you want to maintain all content in the production environment.

5.4 Dedicated Development and Production Environments

Context You're preparing for future changes to your site for which you have already established DEDICATED PRODUCTION AND DELIVERY ENVIRONMENTS (5.3). These changes will not only affect the content, but the underlying software including your templates as well.

Problem **How can you reduce the interferences of software development with content maintenance and publication?**

Example So far we have developed a set of templates as well as several other components for the Technology Transfer Portal. This doesn't mean, however, that there won't be any future changes. Maybe we'll need a new layout one day, maybe there will be additional document types, or maybe we'll want to add more functionality. Either way, changes to the templates and all other components can become necessary.

Changes to the software of the live system will obviously interfere with content production and delivery. The Technology Transfer Portal is not a mission-critical site and therefore has no special availability requirements, but anyway we have to avoid long periods during which content maintenance is impossible, let alone long periods of unavailability.

Forces

All software systems evolve and it's safe to assume that Internet web sites are no exception. Whether the information model changes, whether an organisation wants a new corporate layout, whether demands for additional functionality arise, whether an increased number of users causes new performance requirements — there is a virtually endless list of reasons why changes to a site might become necessary. The extent of these changes varies greatly, ranging from small updates to a complete re-launch.

It's clear that software development must not influence the current content editing process. Content creation, maintenance and publishing must go on undisturbed while new software is developed and tested. Obviously the live software and the software under development must be distinct from each other, as in any other software development project.

Changes to a web site, however, can also involve the underlying information model. In this case the DOCUMENT TYPE HIERARCHY (1.2) needs to be updated, which results in changes to the document type specifications and to the configuration of the CMS repository. You must expect live content and test content not only to be different, but to be differently typed as well. The consequence is that logically different content repositories may become necessary in the context of a web site evolution.

Solution

Establish separate environments for software development on the one hand and for content production as well as delivery on the other. Aim to simulate the live environments in your development environment as well as you can. Make sure to apply versioning to software and content alike.

Setting up the development environment consists of the following tasks:

- Equip the development environment consisting of a central development server and development workstations. The development server requires an installation of the content management system. In addition, you have to provide the necessary tools and mechanisms for software development, such as IDEs and version control.
- Make sure that versioning is applied to all custom components, including not only classes and templates, but also document type model specifications, workflow definitions, and any other configurations. Make a connection between versioning of software (usually done with a configuration management system on the development server) and versioning of the actual content (usually done within the CMS repository).
- Shield your development environment from attacks from the outside by setting up the necessary firewalls, but make sure that deployment mechanisms can pass these firewalls when software updates have to distributed onto the production and delivery environments.

To allow for meaningful tests, the development environment and the live environments must be identical as far as system infrastructure is concerned. This doesn't only refer to the software components installed in each environment, but also includes the underlying operating system, database, load balancers and firewalls.

Ideally this means that your development environment should consist of both a production and a delivery server, so as to simulate the live environment as well as possible. In many practical cases, however, it will be sufficient to use just one installation in the development environment, as Figure 4 illustrates.

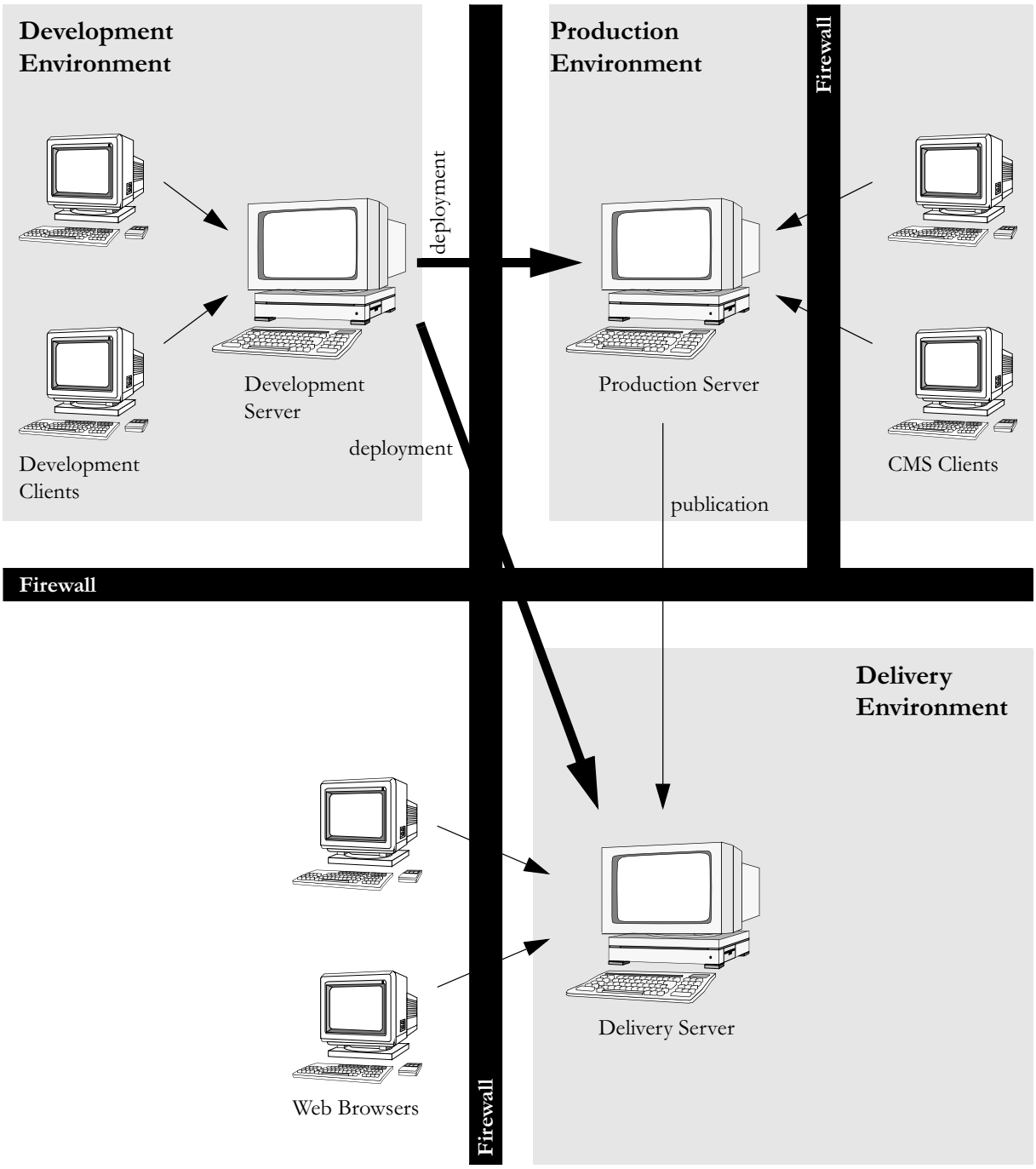


Figure 4 Dedicated development and production environments

Once the tests in the development environment have been completed, you'll have to deploy the software onto the production and delivery servers. This, of course, is the critical moment when the new software version has to go live. There are several strategies that you can apply, depending on the availability requirements placed on your site:

- You can take the site off-line while the deployment takes place. You'll probably need to perform some final tests in the delivery environment too, so the site will stay off-line for a while. Typical time frames range from a few hours to a weekend.
- You can deploy the new software onto some of the servers in the production and delivery environments, perform the necessary tests, but keep the old version online. Once all tests are completed, you can switch from the old version to the new one by changing the web server or the load balancer configuration, so that from that moment on all http requests are answered by servers that run the new software.

Either way, take care to apply a transaction-based mechanism for deploying all custom components from the development environment to the production and delivery environments so that it's possible to roll back to a previous version should the deployment fail, for instance due to problems on the remote side. There are deployment tools available on the market that offer such functionality — you might want to check whether you can use one of those.

If the new version of site assumes a different DOCUMENT TYPE HIERARCHY (1.2) than the old one, a content migration becomes necessary. You'll have to migrate the content in the production environment first and then publish it so that it is transferred to the delivery environment. Content publication has to happen simultaneously with the deployment of the new software onto the delivery environment since only the new software can process the migrated content.

Example Resolved

Since we're not met with requirements for extremely high availability, as well as for cost considerations, we decide to establish a development environment with just one server and several workstations for the Technology Transfer Portal (as suggested by Figure 4). The development server hosts a CMS installation, database, a web server and an application server (or whatever container is required depending on the technology of choice). For testing purposes, the CMS repository in the development environment is supplied with some 'real' content imported from the live system.

Our development environment is protected from attacks by a firewall, but we don't simulate a demilitarised zone in our development environment. We can therefore do a good deal of software development and testing without touching the live system, yet when it comes to deploying new software, a certain amount of tests in the live environment will be necessary.

Benefits

+ Content editing and software development are largely decoupled. Except for the deployment stage, content editors can create and update live content completely independently of any software development activities. Keeping the web site up to date becomes much easier.

+ The dedicated development environment contributes much to the site's testability. The environment can serve as a platform for unit tests, system tests, performance tests and more, resulting in a more accurate and more reliable site.

Liabilities

– The development environment causes additional costs for both software and hardware. This additional cost, however, is the price you have to pay to stay manoeuvrable with respect to content maintenance in the presence of software changes.

– Although the development environment may simulate the live environments to quite some degree, these environments will probably not be exactly the same. Firewall configuration and load-balancing configuration often differ from the development environment to the delivery environment. Even the best tests in the development environment can therefore predict the behaviour in the live system only to a certain degree.

– The deployment process itself cannot be tested completely just inside the development environment simply because it involves file uploads in the live system. You need a fall-back solution if your deployment process fails.

– Different deployment strategies may or may not require you to take the site off-line for a while. If there are changes to the DOCUMENT TYPE HIERARCHY (1.2) and content migration becomes necessary, a short period of unavailability is hard to avoid. But even if the site doesn't have to go off-line, it's likely that no content maintenance will be possible during the deployment stage and while the new version is taken online. Close collaboration between the software team and the content editors is essential here — COORDINATED RELEASE CYCLES (5.5) will help you manage the problem.

5.5 Coordinated Release Cycles

Context

You're planning the future evolution of your site. You have established DEDICATED DEVELOPMENT AND PRODUCTION ENVIRONMENTS (5.4) to decouple software development and content maintenance, and now you're thinking of how to use these environments properly.

Problem **How can you avoid conflicts in the software development and content maintenance processes?**

Example We have established DEDICATED DEVELOPMENT AND PRODUCTION ENVIRONMENTS (5.4) for the Technology Transfer Portal, yet we're aware that when it comes to a software deployment or even a full re-launch of the site a few conflicts between the content editors and the software developers cannot be avoided. On the one hand, the system may have to go down during deployment, and the content editors probably won't appreciate that. On the other hand, software developers must be very careful not to interfere with the live site more than necessary. This requires a good amount of coordination effort.

Forces Maintaining a web site involves different activities, of which software development and content maintenance are the most important ones. These activities are performed by entirely different groups of people — software developers on the one hand and content editors on the other.

Naturally, these activities follow completely different processes. Software development usually includes stages of requirements engineering, design, coding and testing.[2] Content maintenance, however, usually follows workflow processes that include the creation and modification of content, quality assurance and publication. A 4-eye workflow is the most prominent example, though other workflows are possible of course. Unlike the software development process, content maintenance is a continuous process, affecting a web site's appearance more or less permanently.

These processes typically take place in largely independent environments, which generally is a good thing — DEDICATED DEVELOPMENT AND PRODUCTION ENVIRONMENTS (5.4) are necessary to let software developers and content editors work undisturbed by each other. However, the moment a new software release has been completed and should become effective, a connection between the two worlds becomes necessary.

Bringing software developers and content editors together is all the more problematic since usually these two groups of people don't overlap. They are normally recruited from entirely different business units of an organisation. Plans and problems known to one group aren't necessarily known to the other as well.

Collaboration between the two groups is crucial for a smooth site maintenance though. Developers must be aware of additional functionality or changes to existing functionality that the content editors might request, and they must learn of this in due time. Content editors must know about possible interferences of software updates with content maintenance. Everybody must be aware of technical risks that might be involved in changes made to the site, and they must agree on how to react should anything go astray.

2. This list of activities is not supposed to suggest a waterfall model. This is not the place for a discussion of different software development methods. Whether you follow a more traditional or an agile approach, the point here is that the software development and content maintenance processes are clearly different.

Solution **Make sure that all software releases are coordinated by a site manager who is equally aware of the software processes and the demands on content maintenance. The site manager must keep in touch with all stakeholders of the site and ultimately has to coordinate all activities concerned with the site's evolution.**

In detail the site managers' job includes the following tasks:

- develop a strategy for the site's future evolution and come to an agreement concerning this strategy with the content editors
- plan and schedule software changes with the development team
- keep track of and manage dependencies between content and software, especially the need for changes to the DOCUMENT TYPE HIERARCHY (1.2)
- agree on deployment schedules, including the identification of periods during which no updates to the content will be possible
- identify periods during which the site has to go off-line, should this be necessary
- ensure versioning is applied to both software and content, and keep track of which content version goes with which software version, especially in the presence of changes to the DOCUMENT TYPE HIERARCHY (1.2)
- devise possible fall-back strategies should a deployment fail (probably relying on previous versions of the site)

Seemingly unrelated, the software development and content maintenance processes do in fact depend on each other quite a bit. A good deal of coordination by the site manager is the precondition both for a smooth ride through the software release cycles and for undisturbed content workflows.

Example Resolved A relatively small site, the Technology Transfer Portal only has a few content editors and a small development team. Nonetheless, one person has to accept the overall responsibility for managing the site, which includes management of all changes. That person's primary task is to keep in touch with software developers and content editors alike, and to convey the necessary information among both groups.

Benefits

+ The role of a site manager is essential for the site's manageability. The need for changes is recognised earlier, and awareness of future changes is increased. If the site manager coordinates the site's evolution, everybody involved can react more accurately to changing requirements and dependencies on other people's workflows.
+ Because every software deployment is coordinated between both the software developers and the content editors there won't be any unexpected periods during which no content maintenance is possible, let alone any unexpected off-line times. If such periods are necessary, at least the content editors are able to plan ahead.

\+ A fall-back strategy in case of an unsuccessful software deployment increases the site's reliability. Whatever happens, you can always resume the site's most recent version. This version may not include the new functionality you wanted to deploy, but at least it's a consistent version and works reliably.

Liabilities

– The role of a site manager admittedly represents an additional organisation effort, and ultimately increased costs. Someone has to be paid to do the job after all. It's worth it though. Obviously it's work that needs to be done. Plus, if you look at it from a monetary aspect, you can expect the costs caused by an unsuccessful deployment or unexpected off-line times to exceed the costs for proper site management.

Appendix

The following table contains thumbnails of all patterns in the overall collection.

Content Organisation

Domain-Driven Document Type Model (1.1)

How can you ensure that content editors can create, maintain and publish content in an appropriate way?

Introduce a document type model based on domain-driven requirements. The complete set of document types must represent the different information categories known and meaningful to the site's content editors. This way the document types can become the basis for all the site's content management.

Document Type Hierarchy (1.2)

How can you describe the document types in such a way that not only makes rendering possible, but also avoids redundant information?

Extend the document type model to become an object-oriented document type hierarchy. Specify the attributes for each document type and employ association to describe how documents are related. Tailor the document types so that the model becomes normalised and non-redundant. Make moderate use of inheritance to model abstraction.

Decoupling of Content and Navigation (1.3)

How can you ensure that content editors can organise both the content and the site's navigation hierarchy in a straightforward and flexible way?

Decouple the actual content from the navigation hierarchy. Extend the document type model by introducing a special document type for pages. This document type's sole purpose is to embody the web site's navigation structure, while the actual content will be assigned to the existing document types. This way you give content editors the freedom to design content and navigation structures as they see fit.

Configurable Hierarchies (1.4)

How can you support several navigation and content structures simultaneously?

Extend the document type for pages to include conditional links to the actual content and to the child pages. This way you can establish several hierarchies that exist in parallel and can share parts of their navigation hierarchy as well as parts of their content.

Implicitly Linked Documents (1.5)

How can content that changes frequently be maintained, without imposing on the content editors the tedious job of manually linking the new content into the navigation hierarchy?

Introduce a dedicated document type for dynamic lists. A dynamic list does not maintain references to the documents that serve as its list elements, but instead specifies criteria for potential list elements. Later, when the list is being rendered, these criteria will be interpreted to include the appropriate documents into the list.

Type-Specific Validators (1.6)

How can you avoid having documents that contain illegal attribute values?

Specify validators for the attribute types that occur in your document type model. Validators can take different forms ranging from XML schemes to Java code. Make sure that a document can be considered plausible if all validators succeed that can be applied to its attributes.

Workflow-Based Validation (1.7)

How can validators be applied effectively?

Establish two classes of validators. On the one hand, validators that prevent illegal attribute values can be applied during the editing process on the CMS client. On the other hand, validators that check for completeness and consistency have to be integrated into the server-side workflow so that they become effective only upon publication.

Rendering

Document Wrapper (2.1)

How can you avoid that templates are being cluttered with model aspects?

Provide a component for each document type that wraps the access to all document attributes. A wrapper component yields attribute values upon request, but is free to polish these values for proper use in a template. Templates no longer call the system's content API directly, but gain access to documents only through the wrapper components. In their entirety, the wrapper components form an access layer on top of the content API.

Navigation Manager (2.2)

How can you prevent templates from being burdened with the calculation of navigation information?

Establish a component that provides all necessary navigation-related information. Whether they generate navigation elements or hyperlinks to other pages, templates never calculate navigation information on their own but always rely on the navigation manager instead.

Interacting Templates (2.3)

How can you avoid, to a large extent, redundant template code and inefficient rendering?

Define a system of interacting templates. Design the templates to call each other in such a way that they can render the full page, beginning with the page's outermost structure, down to the smallest page elements. The template's call hierarchy will mirror the relationships between document types. Extract all state-dependent rendering into separate templates, and make sure that all state-independent content will be subject to caching.

Template Per View (2.4)

How can you support the definition of different views for the same content?

Define a specific template for each view of a document that is distinctly different from any other views. The primary task of each template is to generate markup for the page element that it represents without specifying any details of its layout. In addition, the template may provide client-side scripting functionality if necessary.

Style Per Layout (2.5)

How can you specify the layout for your site and support different output media at the same time?

Concentrate all layout definitions in style sheets. Each style sheet should contain a consistent set of definitions regarding page geometry, fonts, colours, and the like. Define a separate stylesheet for each view of your site, or output channel, or media type that you have to support.

Careful Use Of Layout Variations (2.6)

How can you allow content editors to fine-tune the layout of individual documents?

If there is a real need for content editors to specify certain layout details for individual documents, you can meet this requirement by assigning additional attributes to the document types in question. These additional attributes represent layout variations from which the content editors can choose. Use this feature sparingly, so as not to make the document types and their associated templates overly complex.

Realistic Browser Assumptions (2.7)

How can you prevent templates from becoming extremely complex due to unknown client-side technology?

Make sure that all web pages you deliver rely only on techniques and mechanisms that are generally accepted and completely unspecific of browser types. On the other hand, make realistic assumptions concerning the availability and the state of the art of the client-side technology.

Searching The Site

Purposeful Search Capabilities (3.1)

How can you ensure that the site's search function meets the users' needs?

Perform a requirement analysis regarding the necessary search capabilities for your site. This includes both the type and the expressive power of search queries that you have to support and a state model that describes the user interaction involved in searching the site.

Decoupled Keywords And Categories (3.2)

How can you make your search function more successful than a mere full-text search would be?

Provide topic-driven categories that each represent some characteristics of some of your site's content. Allow content editors to assign keywords to individual documents and to maintain a mapping that relates each category to a set of keywords. In general it is fine for the categories to overlap. Content matches a certain category if it contains at least one of the category's keywords.

Search Request Controller (3.3)

How can you process the search requests?

Establish a server-side controller that receives all requests concerning the search. This includes search queries as well as requests for sorting, filtering or navigating the search results. The controller does not implement any search functionality but invokes the necessary actions and decides to which follow-up page a user will be forwarded.

Search Engine Adapter (3.4)

How can you smoothly integrate a search engine that yields the desired search results?

Provide an adapter component that wraps the interface of the search engine of your choice. First, this adapter translates search requests received from users into queries understood by the search engine. Second, it takes information about new content and provides it to the search engine in such a way that the search engine can update its index accordingly.

Search Manager (3.5)

How can you maintain the search results that your search engine yields and prepare them for presentation to the users?

Implement a dedicated server-side component that maintains search queries, their parameters and the search results in the session state. This component receives updates when a new query is performed. It makes its information available to the templates that render the search and search result pages.

Personalisation

Role-Based Content Filter (4.1)

How can you can you ensure that the generated web pages contain only content to which the users should be given access?

Establish a mapping that specifies content visibility depending on user roles. Introduce a dedicated component that interprets this mapping and informs about a document's visibility in a given context. A template can rely on this component when it has to determine whether or not it should include certain content in the output it generates.

User Manager (4.2)

How can you introduce user-specific information into the rendering of your content in an efficient way?

Establish a component that maintains the relevant information about a specific user within the user's session state. This includes the user's roles as well as personal information and preferences. Templates and other components call this user manager component whenever they require any user-specific information.

Reasonably Secure Authentication (4.3)

How can you identify the user who's visiting your site?

Employ an authentication mechanism that meets your specific security requirements. You'll have to weigh the level of security that a solution offers against trade-offs such as poor response time.

Role-Specific Templates (4.4)

How can you avoid efficiency problems in the presence of role-based personalisation?

If the degree of personalisation leads to only a small number of different appearances for certain content elements, define dedicated templates for each variation. Each of these templates has to be specific for a certain user role but shouldn't depend on the current user. HTML fragments generated by these templates can therefore be cached.

Verified Registration (4.5)

How can you prevent users from registering false information?

Establish a registration mechanism that includes verification of the user's identity and the personal information submitted. Design the registration mechanism in such a way that the level of security matches the security demands placed on your site.

Profile Maintenance Form (4.6)

How can you reduce the overhead involved in user profile maintenance?

Provide a form that allows registered users to update their profiles. Implement a process that, upon submitting that form, makes the necessary changes in the database.

Deployment And Infrastructure

One Web Application (5.1)

How many web applications should you define for the components that you have developed for rendering content?

Define one web application for all server-side components that are involved in the page generation and delivery for your site. This includes both templates and any service components that you may have developed, as well as components provided by the content management system. It is fine to single out components in separate web applications if their functionality is only remotely connected to page generation.

Legal Request Policy (5.2)

How can you prevent users from retrieving incomplete page fragments, or else from invoking unauthorised requests?

Devise a policy that states which kinds of requests are legal and which aren't. Usually requests for full pages are considered fine, as are requests for page elements that may be integrated into web pages elsewhere. On the other hand, the execution of arbitrary commands or scripts typically leads to unwanted effects, and should therefore be considered illegal. Configure your environment in such a way that it accepts only legal requests.

Dedicated Production and Delivery Environments (5.3)

How can you make your system scalable and at the same time protect it from attacks?

Establish separate environments for content production and maintenance on the one hand and for content delivery on the other. Equip the production environment with the infrastructure necessary to host the workflow processes for content maintenance. Set up a demilitarised zone for the delivery environment and equip it with the infrastructure necessary to process requests from the expected number of users. Except for hardware and security configurations, both environments should be identical.

Dedicated Development and Production Environments (5.4)

How can you reduce the interferences of software development with content maintenance and publication?

Establish separate environments for software development on the one hand and for content production as well as delivery on the other. Aim to simulate the live environments in your development environment as well as you can. Make sure to apply versioning to software and content alike.

Coordinated Release Cycles (5.5)

How can you avoid conflicts in the software development and content maintenance processes?

Make sure that all software releases are coordinated by a site manager who is equally aware of the software processes and the demands on content maintenance. The site manager must keep in touch with all stakeholders of the site and ultimately has to coordinate all activities concerned with the site's evolution.

References

Broemmer 2003

Darren Broemmer. *J2EE Best Practices — Java Design Patterns, Automation, and Performance*. John Wiley & Sons, 2003.

Buschmann Meunier Rohnert Sommerlad Stal 1996

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture, Vol. 1 — A System of Patterns*. John Wiley & Sons, 1996.

Buschmann Henney Schmidt 2006

Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. *Pattern-Oriented Software Architecture, Vol. 4 — On Patterns and Pattern Languages*. John Wiley & Sons, 2006.

Cocoon

The Apache Cocoon Project. "http://cocoon.apache.org".

Dumais 1988

Susan Dumais. "Textual Information Retrieval", in *Handbook of Human-Computer Interaction*. Elsevier (North-Holland), 1988.

Dyson Longshaw 2004

Paul Dyson, Andy Longshaw. *Architecting Internet Solutions*. John Wiley & Sons, 2004.

Farley Crawford Flanagan 2002

Jim Farnley, William Crawford, David Flanagan. *Java Enterprise in a Nutshell*. O'Reilly, 2002.

Flanagan 2002

David Flanagan. *Java in a Nutshell*. O'Reilly, 2002.

Fowler 2003

Martin Fowler. *Patterns of Enterprise Application Architecture*. Pearson Education, 2003.

Gamma Helm Johnson Vlissides 1995

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

Hackos Redish 1998

JoAnn Hackos, Janice Redish. *User and Task Analysis for Interface Design*. John Wiley & Sons, 1998.

Hackos 2002

JoAnn T. Hackos. *Content Management for Dynamic Web Delivery*. John Wiley & Sons, 2002.

Harold Means 2004

Elliotte Rusty Harold, W. Scott Means. *XML in a Nutshell*. O'Reilly, 2004.

Kerth 2001

Norman Kerth. *Project Retrospectives*. Dorset House, 2001.

Kircher Jain 2004

Michael Kircher, Prashant Jain. *Pattern-Oriented Software Architecture, Vol. 3 — Patterns for Resource Management*. John Wiley & Sons, 2004.

Krug 2000

Steve Krug. *Don't Make Me Think — A Common-Sense Approach to Web Usability*.

Loton 2002

T. Loton. *Web Content Mining with Java*. John Wiley & Sons, 2002.

Lucene

The Apache Lucene Project. "http://lucene.apache.org".

Morville 2005

Peter Morville. *Ambient Findability — What We Find Changes Who We Become*. O'Reilly, 2005.

Rockley 2002

Ann Rockley. *Managing Enterprise Content — A Unified Content Strategy*. New Riders Press, 2002.

Rosenfeld Morville 2002

Louis Rosenfeld, Peter Morville. *Information Architecture for the World Wide Web — Designing Large-Scale Web Sites*. O'Reilly, 2002.

Rüping 2003

Andreas Rüping. *Agile Documentation — A Pattern Guide to Producing Lightweight Documents for Software Projects*. John Wiley & Sons, 2003.

Rüping 2005

Andreas Rüping. "Software Architectures for Web Content Management — Best Practices for Enterprises and E-Government", in Andy Longshaw, Uwe Zdun (Eds.), *EuroPLoP 2005 — Proceedings of the 10th European Conference on Pattern Languages of Programs, 2005*. Universitätsverlag Konstanz, 2006.

Rüping 2006

Andreas Rüping. "Software Architectures for Web Content Management — Patterns for Interaction and Personalisation", in Uwe Zdun; Lise Hvatum (Eds.), *EuroPLoP 2006 — Proceedings of the 11th European Conference on Pattern Languages of Programs, 2006*. Universitätsverlag Konstanz, 2007. To appear.

Salton 1989

Gerald Salton. *Automatic Text Processing — The Transformation, Analysis, and Retrieval of Information By Computer*. Addison-Wesley, 1989.

Schmidt Stal Rohnert Buschmann 2000

Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann. *Pattern-Oriented Software Architecture, Vol. 2 — Patterns* for Concurrent and Networked Objects. John Wiley & Sons, 1996.

Sørensen 2003

Kristian Elof Sørensen. "Session Patterns", in Alan O'Callaghan, Jutta Eckstein, Christa Schwanninger (Eds.), *EuroPLoP 2002 — Proceedings of the 7th European Conference on Pattern Languages of Programs, 2002*. Universitätsverlag Konstanz, 2003.

Spring

The Spring Application Framework. "http://www.springframework.org".

Struts

The Apache Struts Project. "http://struts.apache.org".

Turner Bedell 2003

James Turner, Kevin Bedell. *Struts Kick Start*. Pearson Education, 2003.

Vogel Zdun 2006

Oliver Vogel, Uwe Zdun. "Content Conversion and Generation on the Web: A Pattern Language", in Dragos Manulescu, James Noble, Markus Völter (Eds.), *Pattern Languages of Program Design, Vol. 5*. Addison-Wesley, 2006.

Völter Schmid Wolff 2002

Markus Völter, Alexander Schmid, Eberhard Wolff. *Server Component Patterns — Component Infrastructures Illustrated with EJB*. John Wiley & Sons, 2002.

Wellhausen 2005

Tim Wellhausen. "Query Engine — A Pattern for Performing Dynamic Searches in Information Systems", in Klaus Marquardt, Dietmar Schütz (Eds.), *EuroPLoP 2004 — Proceedings of the 9th European Conference on Pattern Languages of Programs, 2004*. Universitätsverlag Konstanz, 2005.

Weiss 2003

Michael Weiss. "Patterns for Web Applications", in *PLoP 2003 — Proceedings of the 10th Conference on Pattern Languages of Programs, 2003*.

Weiss 2006

Michael Weiss. "More Patterns for Web Applications", in Andy Longshaw, Uwe Zdun (Eds.), *EuroPLoP 2005 — Proceedings of the 10th European Conference on Pattern Languages of Programs, 2005*. Universitätsverlag Konstanz, 2006.

White 2005

Martin White. *The Content Management Handbook*. Facet Publishing, 2005.

Wikipedia

Wikipedia — The Free Online Encyclopedia. "http://www.wikipedia.org".

W3C

The World Wide Web Consortium. "http://www.w3c.org".

Hybrid Parser[1]

Jürgen Salecker
Siemens AG, CT SE 2
Otto-Hahn-Ring 6, 81730 München, Germany
juergen.salecker@siemens.com

The *Hybrid Parser* architectural pattern applies to software systems which need to parse documents but are constrained by memory resources and processing power available. The pattern combines the processing advantages concerning execution speed and memory resources of event driven parsers with the programming comfort of a fully-fledged document object model, provided by an object tree parser. An event driven parser is usually by a factor of about 10 faster than an object tree parser but has the significant disadvantage that it is a cumbersome procedure to create an object model out of parsing events. This pattern combines the advantages of both parsing techniques, the solution in a nutshell:
Parsing events are collected and used to construct a parse tree which contains only node location information. Once the addressed node has been reached all following parsing events are used to construct a document object model (DOM) of just this part of the document. This avoids the necessity to hold the complete document in memory, when the focus is only on a part of the document. The extracted part is provided as a complete object model for comfortable processing with a DOM parser by an application.

Context

You need to extract a relatively small amount of information out of a larger (XML, for example) document. This extraction has to be done as efficiently and effectively as possible, both in terms of processing speed and required resources, such as memory and with a minimized implementation effort.

Example

The figure below shows a principle XML document, where the application needs to extract the two DOM sub-nodes trees identified by the two circles.

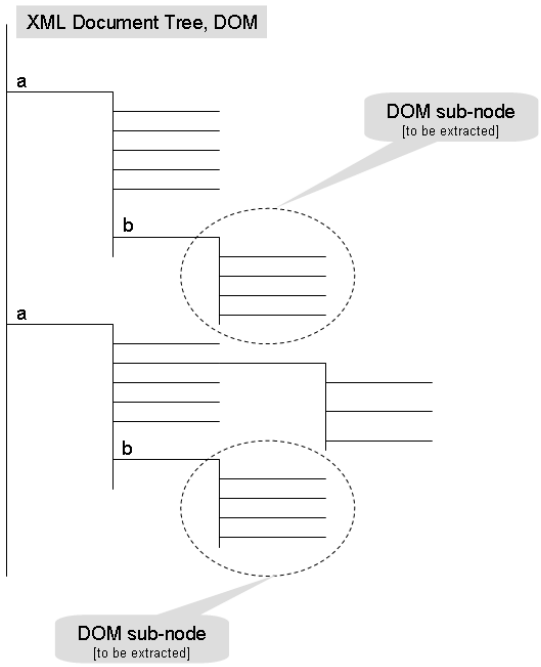


Figure 1, Principle DOM

With an object tree parser (DOM parser in XML technology) the complete document needs to be represented in memory first. Then the application is able to extract the two DOM sub-nodes by using the comfortable DOM API.

In the case an event based parser (SAX parser in XML technology) is used the application is responsible to construct an object model out of the generated parse events, which is a cumbersome procedure. Once the model is complete it is necessarily not a DOM model instead it is an application specific model, i.e. might even contain business logic.

Problem

Let's assume an event based parser is used exclusively to extract the relevant information – the two DOM sub-nodes – shown in the figure above. In this case the processing speed will be sufficient, however the programming effort required creating an object model just out of parse events will be significant. Using an object tree parser instead, will solve the problem to create an object model, but the processing speed is usually not acceptable, at least for larger documents. It can be said that an event based parser is approximately by a factor of 10 faster than an object tree parser.

The following *forces* influence the solution:

- Only a relatively small part of a document is required by the application.
- Provide an effective parsing technique in terms of a minimized implementation effort, because it is a cumbersome procedure to exclusively use parse events to construct an object model of an interesting part of the document. Instead the architect should focus on the business logic and not waste his or her resources "i.e. implementation time" for processing parsing events to create a complete object model.
- The main document to be parsed is too large to be present in memory all at one.
- Provide an elegant measure being able to forward the extracted part of the document to another application(s) as a generic object (i.e. instance of "*org.w3c.dom.Node"*) without containing any application specific logic.
- The extracted object should be easily transformed into other document object models, like XOM, JDOM, Crimson DOM, DOM4J, etc. in order to gain advantages of the benefits of more specific APIs.
- The computer resources in terms of memory and processing power available are limited, as it is typically the case in embedded systems. But similar situations might also occur in enterprise systems.

Therefore the usage of either an object tree parser or an event based parser alone does not produce satisfying results in terms of architectural elegance and resource consumption.

Solution

In favor of a precise description, the solution is explained based on standard and well-known XML parsing techniques. However in principle the solution is generic and can be mapped to other document types to be parsed as well.

The solution in a nutshell: Use a combination of a SAX parser and a DOM builder in order to gain advantage of the processing speed of a SAX parser and the programming comfort of a DOM parser, which provides a fully fledged document object model to the application.

The principle data flow of the hybrid parsing technique is shown in Figure 2 below:

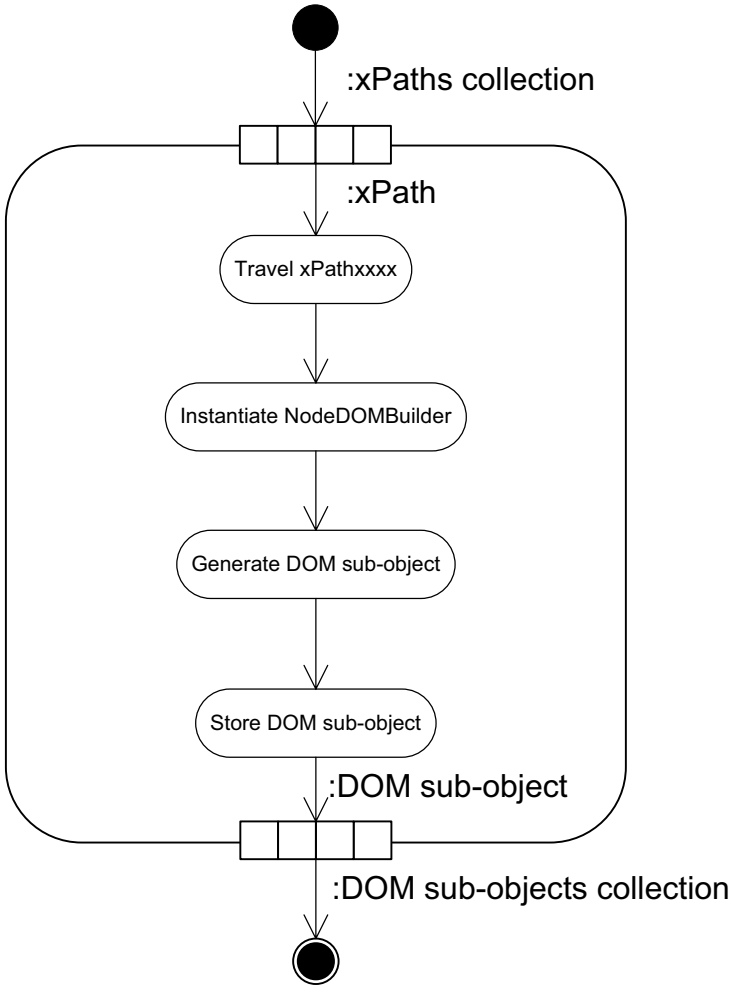


Figure 2, Hybrid Parsing Technique

First define trigger locations (*xpath collections*) for the interesting parts of the document. Then use an event based parser (SAX) to locate (*Travel xpath*) the interesting areas within the document. Once the parser events from the SAX parser match a trigger node (*Instantiate NodeDomHandler*), it feeds all further parse events to a document tree builder. This document tree builder constructs (*Construct DOM sub-object*) a document sub tree out of the token events. The construction will go on as long the document tree builder accesses child nodes of the previous matched trigger node. The tree builder stores the extracted sub-document (*Store DOM sub-object*) once it has processed all child nodes of the trigger node. Finally it returns to the event based parser (SAX), and keeps on parsing.

Again the event based parser is searching for the next match (based on the defined trigger locations) which triggers the same procedure as described above. After the event based parser has finished parsing the complete

document, the collected sub-documents - which are standalone document object models - are forwarded to classes for domain specific processing.

Implementation

The implementation relies on two major principles which are the event based parsing technique implemented by the class *HybridHandler* and the construction of DOM sub-objects out of the parsing events, implemented by the class *DOMBuilder*, a 3rd party software.

The XML processing framework with standard Java 1.5.x has been used for an example implementation; the complete class diagram of the implementation of this pattern is shown in the figure below:

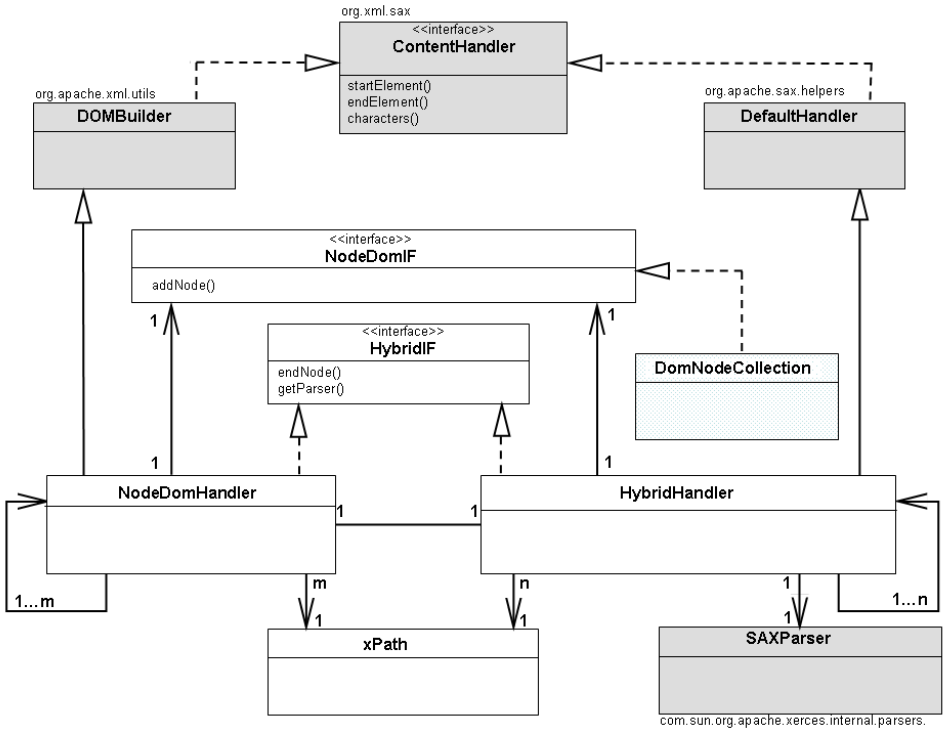


Figure 3, Hybrid Parser, Class Diagram

The slightly grey colored classes indicated 3rd party software from the packages:

3rd party interface

*org.xml.sax.**ContentHandler***

3rd party classes:

*com.sun.org.apache.xerces.internal.parsers.**SaxParser***

*org.apache.xml.utils.**DomBuilder***

*org.apache.sax.helpers.**DefaultHandler***

Software with similar functionality from other 3rd party packages might be used as well by taking into account slight adaptations.

As already mentioned the solution relies on two major principles the "event based parsing technique" together with "DOM sub-object creation out of SAX parsing events".

Event Based Parsing Technique

The principle of this technique is shown in Figure 4, below. A SAX parser calls the method *startElement()* any time a new XML node "<….>" has been detected. The method *endElement()* is called once a closing XML element "</…>" has been detected. The method *characters()* is called for anything detected between the start and the end of an XML node.

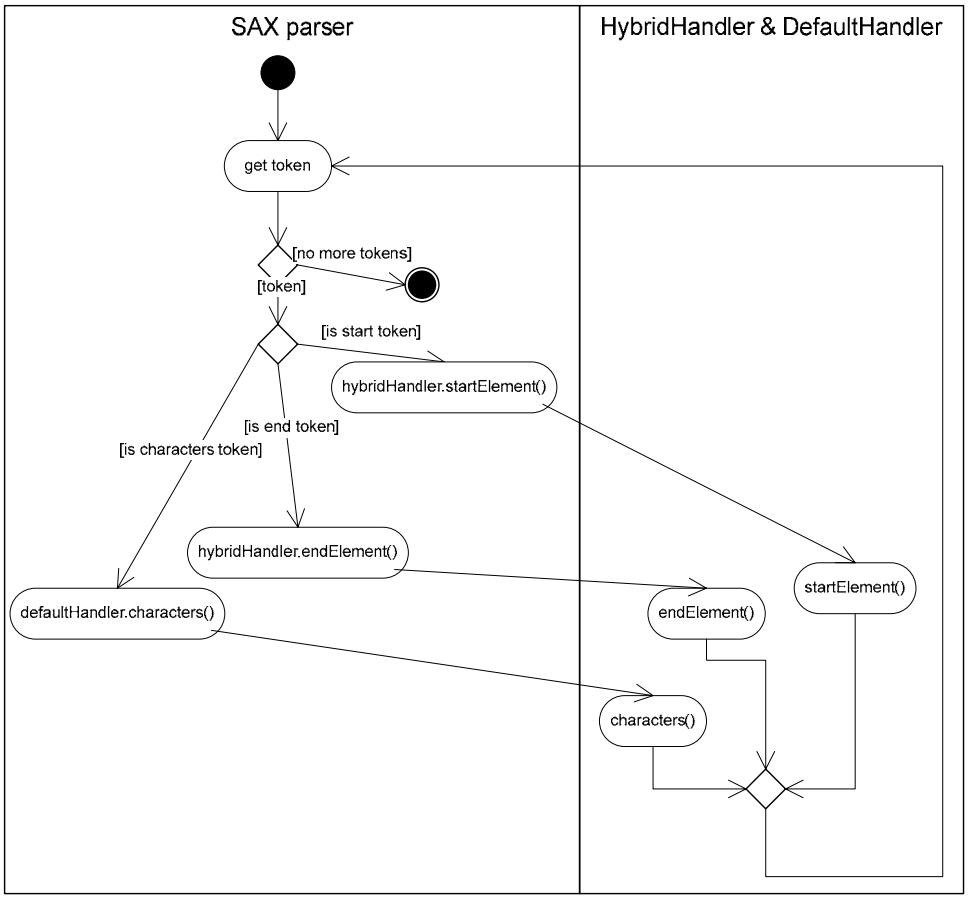


Figure 4, SAX Parsing Principle

The class *HybridHandler* overrides the methods (call-backs) *startElement()* and *endElement()*. The method *characters()* will be handled by its super class, the *DefaultHandler* (3rd party software).

The methods *startElement()* and *endElement()* are used exclusively to construct XML tree location information (xpath). The XML node name

(including namespaces) is used to calculate the location information (xpath) of the current parse location. All other information provided by those two methods (XML attributes) is ignored, i.e. not relevant for this pattern.

If the calculated location information (xpath) matches with the xpath definition provided by the class *xPath,* the *HybridHandler* will instantiate the class *NodeDomHandler* and then forward the SAX parsing events to this newly created object.

DOM sub-object Creation

The class *NodeDomHandler* together with its super-class *DOMBuilder* implements the second principle of this pattern, the creation of a DOM sub-object. It uses the forwarded SAX events to construct a complete DOM sub-objects by including all details, XML attributes and name spaces. It uses the same instance of the SAX parser as the class *HybridHandler*.

NodeDomHandler overrides the methods *startElement()* and *endElement()* (call-backs from the SAX parser) and within those methods it calls its super class *DOMBuilder* for DOM sub-object creation, outlined below:

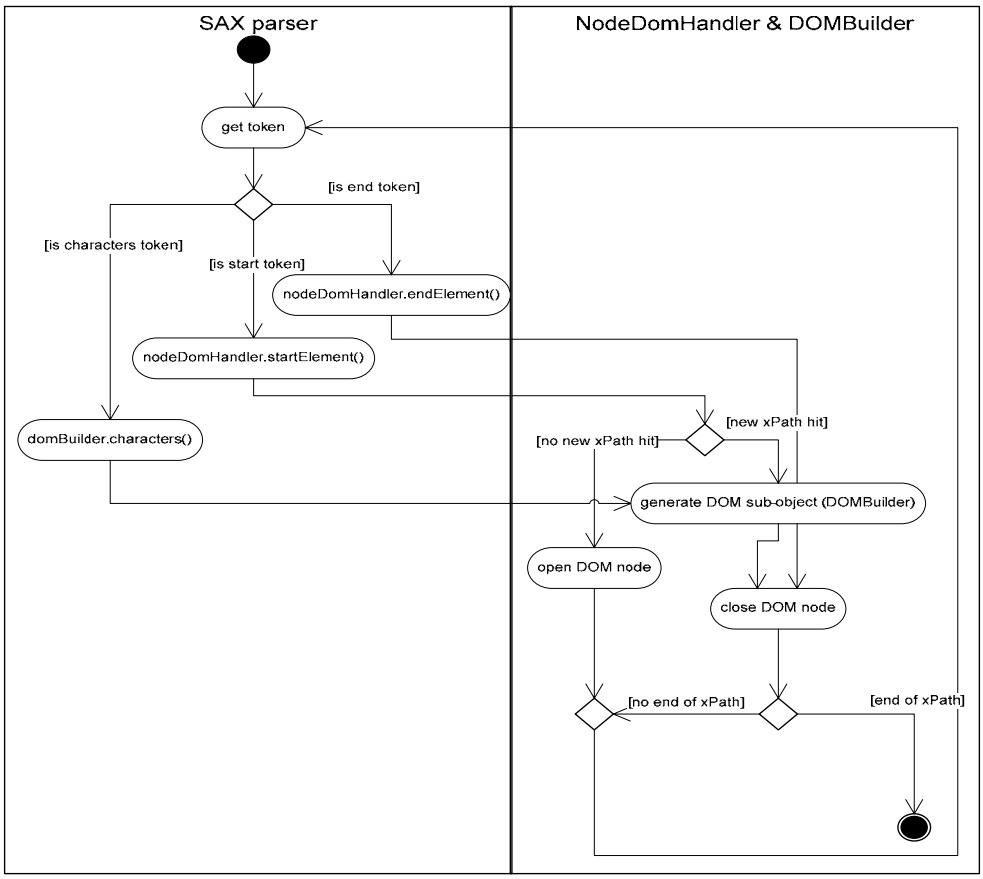


Figure 5, DOM sub-object Creation

The class *NodeDomHandler* calculates as well document location information (xpath) in the same way as *HybridHandler,* but with the exception: If the calculated location information matches with the xpath definitions provided by the class *xPath* it will instantiate itself recursively. An example XML document for this case is shown in the figure below:

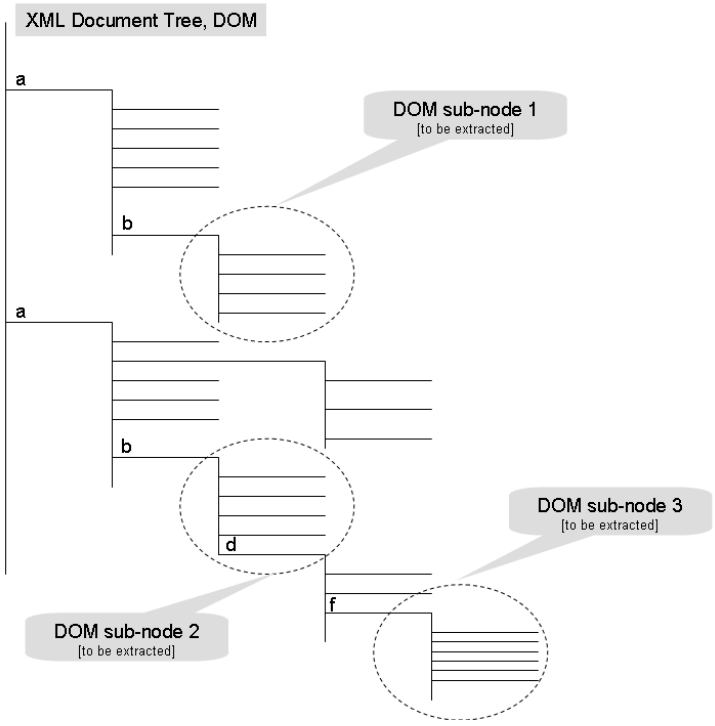


Figure 6, Extracting XML Child Nodes

This provides a comfortable way to represent an XML tree as a tree of instantiated DOM sub-objects. In the example document model shown above (Figure 6) the DOM node three "./d/f" will be extracted as a DOM object from DOM node two. DOM nodes one and two have the same xpath "/a/b" but have different child nodes.

Once the *NodeDOMHandler* has reached its end condition (the same leaf level in the parse tree as it has been instantiated) it calls *endNode()* which adds the constructed DOM object to the user defined class via the method *addNode()* implemented by the class *DomNodeCollection.* Then it returns the SAX events back to its parent object. Its parent object could be either another instance of the same class *NodeDomHandler* or an instance of the class *HybridHandler.* Returning SAX events back to the previous initiator is initiated by the method *getParser().setContentHandler()* from interface *HybridIF*.

The class *DomNodeCollection* represents a class which holds the extracted DOM sub-object collection. This is the only class which might contain business specific code and is therefore not part of this pattern. It is shown in the class diagram (Figure 3) for the sake of completeness.

The class *HybridHandler* might be instantiated recursively in case XML documents are linked together via Xlink definition (see Figure 7 below). In those cases the *HybridHandler* has to be launched at the XML root document.

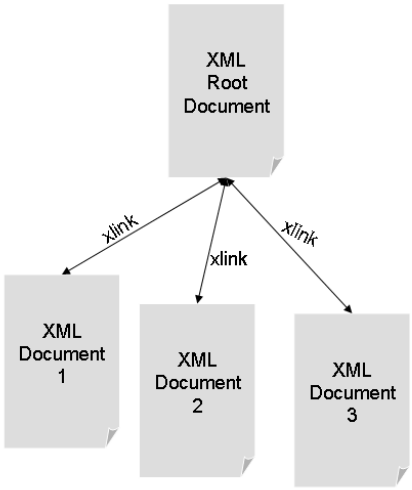


Figure 7, Recursive XML Document Processing

Known Uses In the Q-Marine system - a seismic recording system - from Schlumberger the pattern is used for managing the massive amount (well above 15000) of digital sensors (small embedded system) in an efficient way. At Siemens it is currently being implemented for managing the vast amount of information from software change information (tasks from a configuration management system) in an efficient way.

Consequences The pattern provides the following benefits:

- Light weight parsing of even large documents by resource constraint systems, both in terms of processing speed and memory resource required.
- Combing the advantages of event based parsers with object tree based parsers.
- Avoid the cumbersome procedure of having to implement a complete object model out of parse events.

The pattern provides following liabilities:

- If the application requires more or less the complete document and enough memory is available one might consider using an object tree parser exclusively.
- This pattern does not provide a direct binding of the XML document to the programming language in usage, because the document object model has to be queried via the API provided by the document object model (DOM).

Credits This pattern has gone a long way to reach the point where it is now. As I decided to use XML in a development project somewhere in 2002 at Schlumberger/Norway, I searched the Internet for advice about how to parse large XML documents as efficient and effective as possible. In a forum (I forget the name of the original author) there was a post with the following four advices concerning the efficient processing of large XML documents:

- Define trigger locations for interesting parts of an XML document.
- Use an event based parser (SAX parser) to locate those locations.
- Use a parse tree builder (DOM builder) to extract the defined location as a DOM sub-object.
- Use a tree parser (DOM parser) to provide the binding to the programming language.

The core of the pattern is captured in those four statements. Also thanks to Kevlin Henney who basically invented the name of the pattern (in Q1/2003) once I explained the principle to him.

My first shepherd (EuroPlop 2005) pointed out that this pattern might be more generic as just to being applied for XML document processing. At this time I did not really believe, but at this comment got repeated by shepherd number three (Uwe Zdun, VikingPlop 9/2006), finally I was convinced. Shepherd number two (Peter Sommerlad, EuroPlop 7/2006) did provide very valuable support concerning the clarification of the essence of this pattern.

The writer's workshop at VikingPlop 2006 composed of: James Siddle (SAP), Cecilia Haskins (NTNU), Juha Pärssinen (VTT), Met-Mari Nielsen (Runestone Game Development), Pavel Hruby (Independent), Klaus Marquardt (Dräger) did provide very useful suggestions in order to increase the quality of the paper even further. The UML diagrams became significantly clearer with the help of my colleague Silvano Cirujano-Cuesta, a UML specialist, at our Munich office. Finally my colleague Jürgen Schmitt initiated important last minute changes, concerning the correct description about XML child node processing.

Patterns in Other Fields

Patterns for Tailoring E-Learning Materials to Make them Suited for Changed Requirements

Birgit Zimmermann, Christoph Rensing, Ralf Steinmetz

0 Introduction

Creating appropriate E-Learning material is a costly task. Re-use would help to lower those costs. But if you want to re-use existing material you need access to high quality material that can be re-used. By establishing a marketplace for high quality E-Learning materials the project Content Sharing (http://www.contentsharing.com) aims to solve the problem that often smaller companies have no access to such materials. This marketplace will make E-Learning materials available to all kinds of companies for use and re-use. But often the re-used materials do not completely fit to the new usage scenario. In those cases the materials have to be adapted to the changed requirements.

There exist lots of different kinds of such adaptations (e.g. adaptations to a changed corporate design, terminological adaptations, or content translation). This means that several aspects must be considered (layout, didactics, linguistics, technology). In addition various different formats are used (such as HTML, PPT or Flash), often within one course. Unfortunately, many authors do not have the knowledge to perform all the adaptations needed. Therefore a tool offering support to authors in performing E-Learning material adaptations would be useful.

To design such a tool, we analyzed which and how adaptations are done by experts [4]. We found that there are common ways how to do the adaptations. Some times adaptations can be done in an automated way (e.g. changing the design of a course). Some times it is hardly possible to offer a good automated support, as the adaptations are done based on human cognition and experiences (e.g. adaptation to a changed learning objective). But in all cases we found that there exist guidelines and best practices how those adaptations can be done. Therefore we decided to write a pattern language for the adaptation of E-Learning material.

In this paper we present a first version of solutions we have mined. We present them as five patterns that form a pattern language for E-Learning material adaptation. The aim of this language is to collect expert knowledge in performing single adaptation processes and to make it available to people who have to perform several adaptation processes. Those people have a certain basic knowledge that is needed to perform the adaptation processes. For example they may know how to change the size of an image. But they may not be experts in all the adaptation processes they have to execute.

Talking to experts we got a list of 15 adaptations (listed in table 1) performed by those experts. We also asked the experts to describe how they proceed in performing the adaptations. When analyzing the answers we got from the experts we found that adaptations cover three areas of changes:

- Changes in the *layout*,
- Changes in the *content*, and
- Changes with a more *technical background or purpose*

Each of the 15 adaptations belongs at least to one area. Adapting material to a changed design for example changes the layout of the material, changing the terminology changes the content. But some adaptations belong to several areas. For example the transformation into another format is mainly an adaptation with a technical background. But it also changes the layout, e.g. by resizing images that are not suited for the new format. With our pattern language we want to cover all areas of changes as all of them might be needed to make re-used material perfectly suited to changed requirements. The following table shows all adaptations and their assignment to the area of change they mainly belong to. The adaptations covered by the patterns in this paper are written in a bold font.

Layout	Content	Technical background or purpose
• **design** • **printability** • screen resolutions • accessibility	• **translation** • learning objective • terminology • degree of interaction • semantic density (see explanation below) • learning strategy • difficulty of the course • duration of the course	• transformation into several formats • end devices • bandwidths

Table 1: Areas of adaptations.

Explanation to semantic density: "The degree of conciseness of a learning object. The semantic density of a learning object may be estimated in terms of its size, span, or --in the case of self-timed resources such as audio or video- -duration." [IEEE Learning Technology Standards Committee: IEEE Standard for Learning Object Metadata 1484.12.1., 2002]

In addition we found that there exist certain connections between adaptations: Two adaptations *are connected* if the execution of the first adaptation probably leads to the execution of the second adaptation.

Example: If you change the semantic density of a course you should also check if you have to change the difficulty of this course. The connection can be very close or looser. E.g. the connection between semantic density and difficulty is close as a course with a high semantic density is normally more difficult then a course with a low semantic density. So changing one of those two items probably leads to a need for a change in the other one as well. Between design adaptation and terminology adaptation there is a loose connection. Design adaptations are often needed if one company decides to re-use the course of another company with a different corporate design. The companies might also have specific terminologies which results in a need to adapt the terminology.

The following figure shows the connections between the three adaptations that are described by the patterns in this paper. Each adaptation is described by one pattern which is named according to the adaptation. The arrows between two adaptations mean that if the first adaptation process is performed this probably leads to a need to perform the second process as well. The patterns describe how to perform the adaptations.

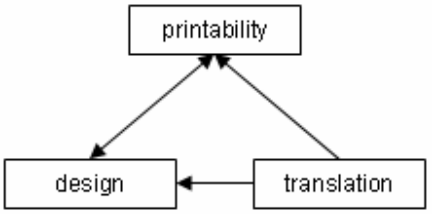


Figure 1: Connections between adaptations.

Our pattern language shows how to adapt E-Learning material to make it suitable for a new scenario of usage. There may be other pattern languages related to our language, like Ian Graham's "WU pattern language" [1] or Vogel and Zdun's pattern language for "Content Conversion and Generation on the Web" [3]. But up to now no pattern language for adapting existing E-Learning material to changed requirements exists.

There exist two kinds of patterns:

1. First we formulate 15 *adaptation patterns*, one pattern for each adaptation. Those patterns describe how to perform an adaptation. Whenever someone wants to adapt E-Learning material to make it suited for changed requirements he / she finds the description what to do in those patterns. (This paper contains three adaptation patterns: “Design Adaptations”, “Printability” and “Translation”.)

2. Second we formulate what we call *supporting patterns*. During the execution of an adaptation process there are typical problems you might run into. Those problems are described by supporting patterns. Several of those problems occur in more than one adaptation. For example there are many adaptations that replace parts of a text. The new text parts can have a different size than the old ones. If it is important that the new text parts keep the length it is necessary to correct the text parts that differ in their length. A description of how to solve this problem can be found in the supporting pattern “Correct length of text blocks”. Therefore supporting patterns describe how to solve problems that typically occur during performing adaptations. (“Correct Length of Text Blocks” and “Correct Arrangement of Elements” are the supporting patterns described in this paper.)

Problems can also occur during the execution of a solution described in a supporting pattern. To solve the problems that come up here again supporting patterns are used. If a supporting pattern does not need any other patterns to support its execution we call this a final supporting pattern.

As stated before our target group is not the group of people being experts in performing one special adaptation (like a translator is expert in performing translations or a designer is expert in changing the design). We aim at those people who have to work with adaptations but are not experts in performing them. To understand our pattern language, only a basic knowledge in adapting E-Learning material is needed. But to adapt existing material in a useful way, persons who do adaptations should have knowledge in the topic the course covers.

1 Re-Using existing E-Learning Material *

Intent: Change existing E-Learning material to meet new requirements.

Context: Creating appropriate E-Learning material is a costly and time consuming task. One possibility to make the production of E-Learning material less expensive is to re-use existing material. This is particularly relevant for smaller companies that cannot afford to create custom material tailored to their particular needs. Re-using material can either mean that you take parts of an existing course, e.g. a good example, and add them to a new course. Or it also can mean that you re-use a whole course. However, re-using existing material is not enough since often the existing material does not completely fit the new usage scenario. As a result of this, the material has to be adapted to this new scenario.

Problem: You want to adapt existing E-Learning material to changed requirements. What do you have to do in order to achieve material that fits the new requirements?

Forces:

- You have to create new E-Learning material.
- You re-use a whole course or only parts of it, as they (at least partially) fit to your needs.
- The re-used course or the re-used parts do not completely match to the new scenario of usage. But it is cheaper to adapt them then to build them from scratch.

Solution: You start by defining your requirements. Then you decide which existing material best fits your needs. To achieve a 100% match between your requirements and the re-used material it might be necessary to adapt the existing resources to these requirements. How costly an adaptation is depends on the tools you have supporting you in performing the adaptation, and on the knowledge you have for this adaptation. You have to consider 15 possible adaptations:

- design: the (corporate) design of the material has to be changed
- printability: you need a version optimized for printing it
- screen resolutions: you need versions optimized for several screen resolutions
- accessibility: the material has to me made available to disabled persons
- translation: the material has to be translated into another language
- learning objective: the learning objective has slightly changed (if major changes should be done to the learning objective, you have to create new material)
- terminology: the terminology used in the material has changed
- degree of interaction: you want to change the degree of interaction
- semantic density: you have to adapt the material to a changed semantic density
- learning strategy: the material has to be adapted to a changed learning strategy
- difficulty of the course: The degree of difficulty of the material has to be changed
- duration of the course: the material should change its duration
- transformation into several formats: you need several file formats (e.g. PDF and HTML)

- end devices: users will access the material using different devices (PDA, note book,…)
- bandwidths: users download the material via different bandwidths. The material should be optimized to achieve the best possible download rates for several bandwidths

Known uses: This pattern is based on the experience of several people working in the area of E-Learning material production.

Consequences:

Positive:

- You save time and money.
- You get E-Learning material that completely fits your needs.

Negative:

- You have to do the adaptations.
- You have to take care not to forget an adaptation. Otherwise you would get material that is not really suited for your needs.

Related patterns: Not known

Connected Patterns:

- None

Used Patterns:

- You adapt E-Learning materials to changed requirements with adaptation patterns such as TRANSLATION (for different languages), DESIGN ADAPTATION (for different designs) and PRINTABILITY (to get a printable version).

2 Design Adaptation *

Intent: Adapt the design of materials to match incoming requirements.

Context: As for many kinds of content the design is very important for E-Learning content. Therefore you should always take care of a design matching all requirements. If there is a change in design requirements it is necessary to adapt the course to the new requirements. There are several reasons for a change in the requirements, e.g. if a course was originally designed for one company and should be re-used in another company or if the style guide of a company changes.

Problem: You want to adapt your course to changed requirements concerning the (corporate) design. What do you have to do in order to achieve a design that fits the new requirements?

Figure 2: Example for a design adaptation.

Fig. 2 shows a course: First the original version, and then after adapting it to a changed corporate design.

Forces:

- E-Learning courses are normally designed by following a style guide. If this style guide changes for some reason the design of the course has to be adapted to the new guideline.
- The design normally consists of many items, like logos, background images and colors, fonts etc. To adapt the design all elements have to be considered.
- If a style template is used you can change this template (e.g. CSS for HTML or slide master for PPT).

- If no style template is used you have to change the design by changing element by element, page by page and file by file.

Solution: A design adaptation starts by replacing graphical elements that do not meet the requirements (e.g. logos). Therefore you decide for each graphical element if it is conform to your requirements. If it is not you replace it by a conform element. If the new graphical elements have a different size compared to the original ones it might be necessary to resize them. Depending on the file format of the materials the way how you replace the elements is different. E.g. in HTML you replace the target of the element's tag, whereas in DOC you delete the old element and insert a new one.
There might be some graphical elements that occur in places where no elements at all are allowed to occur. You should check for those elements and if you detect some you must delete them.
If you need additional graphical elements in places where no element is provided you have to add those elements. Keep in mind that this has effects on the arrangement of all elements. In the last step you rearrange all elements that are not placed correctly.
Style guides normally define rules how to design the whole layout. If the style guide changes you have to adapt the design accordingly.
If you want to use the course in another company it might be also necessary to change the company name. Be careful if the new name has a different length then the old one. This might has a negative effect on the look of the text blocks, where the name has been changed.
There is no strict order in executing the steps mentioned so far. However the order proposed here seems to be useful. The step "Rearranging text parts and images" should always be executed as the last step. When executing this step you should check that all elements are positioned correctly. All other steps might influence the arrangement of elements. E.g. if a logo is deleted instead of replacing it, this leads to a change in the arrangement of the other elements as well.

Steps needed to execute the solution:

1. Replacing graphical elements
2. Deleting graphical elements
3. Add additional graphical elements
4. Performing changes according to style guides
5. Changing company naming
6. Rearranging text parts and images

Known uses: This pattern is based on the experience of experts in branding and corporate design from several companies, e.g. SAP, as well as on our experiences in changing the design of E-Learning content.

Consequences:

Positive:

- The course is adapted to the required (corporate) design. This generates the required look and feel for this course.

Negative:

- In the rare cases where no style guide is available the adaptation is hard to execute and might be incomplete.

- If you have done adaptations that provide an additional version of the course for parallel use (like translation or printability) you should perform the design changes for these versions as well.

Related patterns: Not known

Connected Patterns:

- Terminology (Often the target group has changed if a layout change has to be executed. A change in the target group might also require an adaptation to a changed terminology.)
- Printability (If you have changed a version that is optimized for printing, you should check that none of your changes conflicts with printability.)

Used Patterns:

- "Correct arrangement of elements" used by "Rearranging text parts and images"
- "Correct length of text blocks" used by "Changing company naming"

3 Printability *

Intent: Printing E-Learning materials appropriately.

Context: Most E-Learning materials are designed to access them via a browser. Often materials are without any provision for printing them in a suitable way. For example, a course may allow a student to exercise an online test, but not to print the results. But often users want a printable version of the materials. As Ian Graham says: "Navigability and aesthetics conflict with printability…" [1]. Therefore you should provide two versions: the original one optimized for output on a screen, and a second, separate version that is optimized for printing. The user can then print the material if he / she wants to do so.

Problem: You want to provide a separate version of the material that is optimized for printing. How do you optimize the new version for printing?

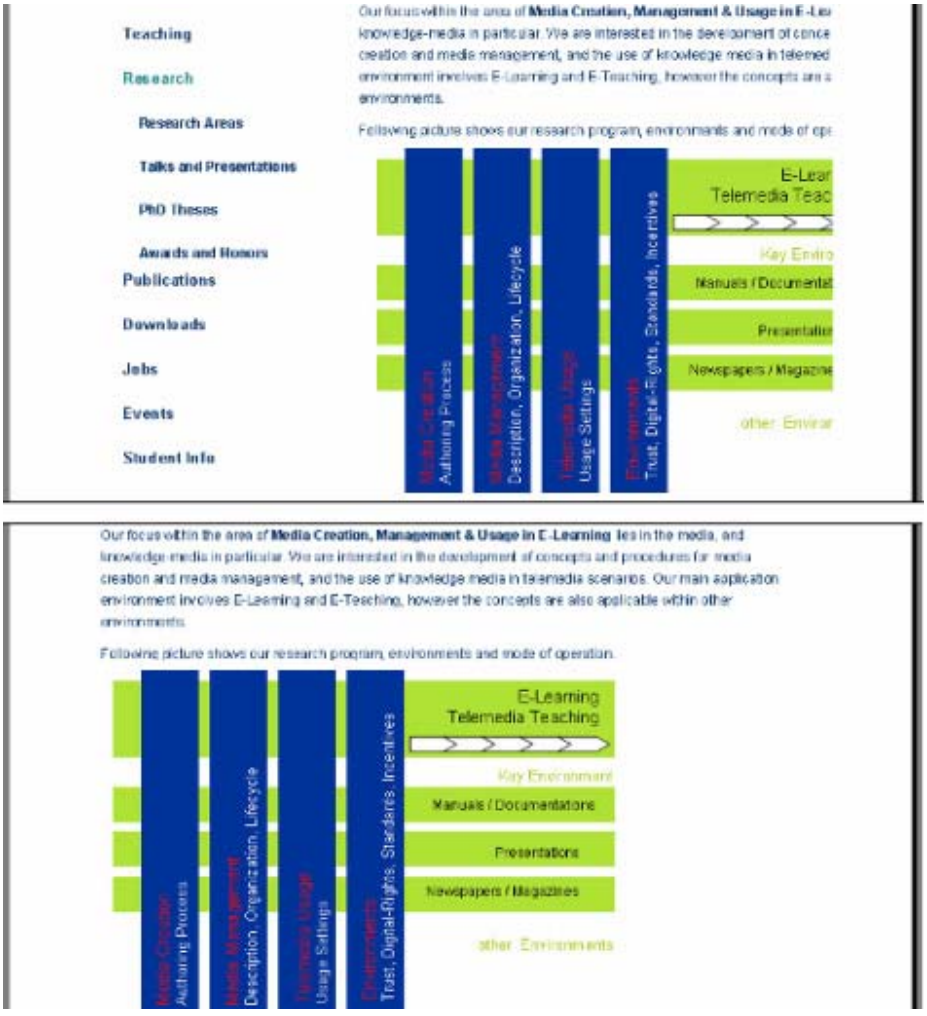


Figure 3: Example for a printability adaptation.

Fig. 3 shows how the research department KOM solved the printing problem. The first part of the image shows a small section of the print preview of a part of KOM's homepage. You can see that some elements on the right hand side are cut off. To overcome this problem KOM offers a second version of their website that is optimized for printing. This is demonstrated in the lower part of the figure. You can find the same problem and the nearly same solution in the area of E-Learning material.

Forces:

- Adaptation to achieve a good printability is not needed for all formats. Page based formats like PDF or DOC that have been created to be printed later on, do not need to be adapted.
- Your material is not optimized for printing, but for access e.g. via a web browser or a special player for E-Learning courses.
- When printing the material some elements might be cut off.
- Sometimes formatting data like CSS is available and can be used for optimization, sometimes this is not the case. But often it is not enough to "only" change this CSS file, you also have to take care for images if they are used.
- When printing some elements might be separated in a way that downgrades readability and understandability.
- You need a version optimized for printing in addition to the version optimized for viewing it on a computer screen.

Solution: In general you should create a new printer friendly version of your material. This means you have to create a second version of your material that is provided in addition to the original material. To achieve a good printability for this new version you have to resize all elements that cannot be separated in a way that they match with a printed page (e.g. images that are too big). You have to do the same for all elements that can be separated (e.g. text boxes or tables). You should remove all colors that cause negative effects when being printed (e.g. yellow font color, or white font on a black background). You should check for elements that contain important information but are not printed. Those elements should be added in a printable version as well. And you should correct all page breaks that reduce readability. The order of the steps is partly fixed (the first two steps and the next two steps belong together). It is useful to follow the order proposed here to achieve a good result.

Steps needed to execute the solution:

1. Detecting non separable elements (like images) not fitting to targeted page size
2. Matching non separable elements to targeted page size
3. Detecting separable elements (like tables) not fitting to targeted page size
4. Matching separable elements to targeted page size
5. Checking colors not suited for printing
6. Checking for elements that are not printed (like backgrounds)
7. Correcting page breaks

Known Uses: You find lots of examples where a separate version of a resource optimized for printing is provided, e.g. company homepages like the one of the company SAP mentioned above.

The problem of printer friendly versions is considered often in the hypermedia area. In this area other patterns were developed to solve this problem. E.g. Lyardet and Rossi [2] have written a pattern called “Printer Friendly” and Graham [1] wrote another pattern “Separate print pages”. This pattern uses ideas from their work.

Consequences:

Positive:

- You have a version that is optimized for printing.

Negative:

- Having two versions, one optimized for viewing on a screen and one optimized for printing, causes more effort in keeping both versions current.
- If you have done adaptations that provide for additional versions of the course for parallel use (like translation) you should create a printable version of those versions as well.
- This adaptation provides a print version for each page, if you need a print version for the whole course you should additionally convert the course to a format suited for printer output (e.g. PDF).

Related patterns: “Printer Friendly”, “Separate print pages”

Connected Patterns:

- Design Adaptation (If you match several elements to the targeted page size or correct page breaks, this might influence the arrangement of the elements in a way that is not conform to the design guide requirements.)

Used Patterns:

- “Correct arrangement of elements” used by “Matching non separable elements to targeted page size”, “Matching separable elements to targeted page size “, and “Correcting page breaks”

4 Translation *

Also known as: Multilingual Content Production

Intent: Provide content in a different language.

Context: Normally E-Learning material is first designed in one language. If versions in another language are needed the original version has to be translated.

Problem: You want to translate E-Learning content from one language to another language. Which process has to be performed to achieve a translated version?

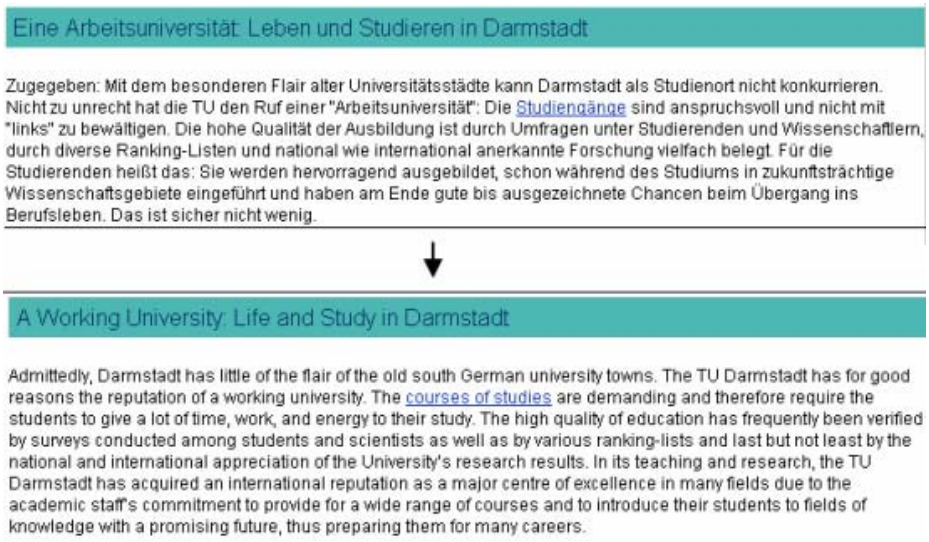


Figure 4: Example for multilingual content.

Fig. 4 shows the first paragraph of an introduction to the University of Darmstadt. The upper part shows the German version; the lower part shows the translated English version.

Forces:

- Your content is provided in a language that is not understood by your target group.
- To provide a version understandable to your target group you need to translate the material.
- You know in which source languages the content is written and to which target language it should be translated.
- You need to know whether only parts of the course or the whole course has to be translated.

Solution: To create a translated version you have to provide the content to the translator. Caution: You should always provide as much information to the translating person as possible. This allows for a better quality of the translation. (If for example the translator has only a few sentences to translate but not the whole text he or she probably may have difficulties to understand the correct meaning of the sentences depending on the context.) In addition you should make available domain specific information to the translator. In most domains a specific terminology is used that has to be regarded during translating the content.
The translator translates the content. If the length of the text is important it has to be checked during translation that the translated text stays within the allowed length. This means that you have to provide the information if the text size matters to the translator. If the text size is of importance the translator also needs information about the allowed text size and possibilities how to correct the size, e.g. if certain abbreviations have to be used. Based on this information the translator then has to check and if necessary correct the length of the text. If only parts of the material have been translated you or the translator have / has to add or to replace those parts. (Adding might be necessary if you want to keep the original part, e.g. a definition, and want to add the translation.) After adding or replacing the translated elements you or the translator should check if the arrangement of the elements is still according to your requirements. (This is not necessary if you replaced text parts by new texts with the same length.) The order of the steps needed for translation is fixed. This means that you should execute the process in the order as it is described here to achieve an optimal result.

Steps needed to execute the solution:

1. Making elements that need to be translated available for translation
2. Translating content
3. Checking for correct length of text
4. Resizing texts with a wrong length
5. Adding translated elements
6. Replacing translated elements
7. Re-arranging elements that do not fit the requirements

Known uses: We talked to people in several companies, e.g. SAP, providing E-Learning materials in several languages. In addition we talked to several translators. We found that the solution described above is accepted as best practice.

Consequences:

Positive:

- The translated text elements are now available in the desired language.
- If you decide that the length of the text in both versions is not important you have no problems with texts being too long or too short.

Negative:

- If the length of the text is not important you have two different versions with respect to the layout of the course.

- If the length of the text is important the translator has to take care of formulations that match the length of the original text. This might cause many abbreviations or formulations that have slightly different meaning then the original text. In addition the readability might be decreased by many abbreviations.
- Providing a translated version of your course causes a higher maintenance effort as later changes in the original course also have to be translated.
- You have to translate each parallel version of your course (e.g. a print optimized version) as well.

Related patterns: Not known

Connected Patterns:

- Printability (After translation of material that has been optimized for printing a further optimization for printing may be necessary.)

Used Patterns:

- "Correct length of text blocks" used by "Resizing texts with a wrong length"
- "Correct arrangement of elements" used by "Re-arranging elements that do not fit the requirements"

Remark: At the moment only a few supporting patterns are available, but translation needs lots of supporting patterns as it is a complicated task. This will be taken into account in future versions of this pattern language

5 Correct Arrangement of Elements

Intent: Get a correct arrangement of the elements contained in your course.

Context: There are several adaptations that change the arrangement of course elements, like text blocks or images. In some cases you need to re-arrange the elements.

Problem: You have to change the arrangement of some elements in order that it fits to your requirements. How can you execute the re-arrangement?

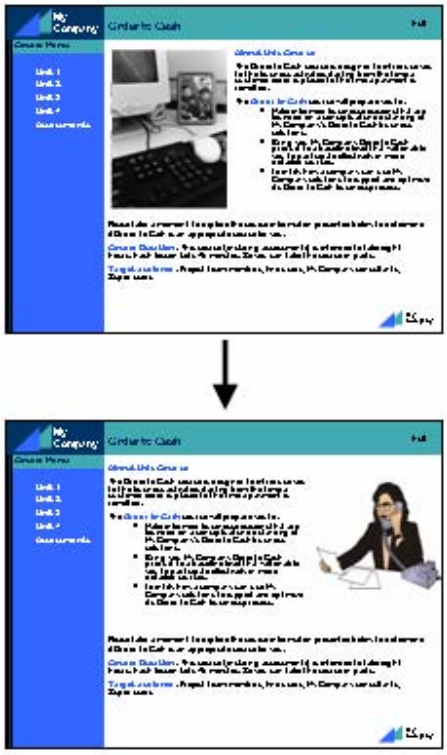


Figure 5: Example for re-arranging elements.

Fig. 5 shows a course page where a photograph has been replaced by an image. In addition the image had to be placed on the right hand side of the page. This made a re-arrangement necessary.

Forces:

- The arrangement of some elements in your material has changed for some reason (e.g. deletion or insertion of graphical or textual elements or replacement by other elements with a different size).
- The new arrangement does not comply with your requirements, e.g. given by a company style guide.

- The guidelines concerning the arrangement of course elements have changed and you have to adapt the material to the new requirements.
- Or the requirements how to arrange elements have changed.
- You must re-arrange the elements to achieve an arrangement compliant with the requirements.

Solution: You have to re-arrange the elements in a way that they comply with the requirements, e.g. given in a company style guide. There are several reasons why the arrangement of the elements is not correct (see forces).

If the arrangement has changed because you have replaced an element you can try to change the size of the new element. If for some reason (e.g. loss of quality) this is not possible or if you have added or deleted an element you can try to resize the surrounding elements. If you resize elements you should always take care not to decrease the quality.

If you cannot resize the elements or if the requirements on the arrangement of elements have changed you have to rearrange the elements. This means that you have to check how the elements can be arranged to achieve a result that supports a good readability and understandability and that complies with the requirements.

Known uses: This pattern is based on our own experiences with changing the arrangement of elements in E-Learning courses and on company guidelines describing the arrangement of elements in different formats.

Consequences:

Positive:

- The elements are arranged correctly.

Negative:

- Depending on a potential resizing of elements their quality might have decreased.
- The new arrangement might be not as good as the original one, but at least it should be better than before the re-arrangement.

Related patterns: Not known

Used Patterns: None

6 Correct Length of Text Blocks *

Intent: Get a correct length of the text parts contained in your course.

Context: There are several adaptations where you replace text parts by new text parts, e.g. changing a companies name, changing the terminology or translating content. If the new text element has to keep the length of the original element you have to change the new text element in a way that it gets the correct length.

Problem: You have to correct the length of text parts. How can you execute the correction?

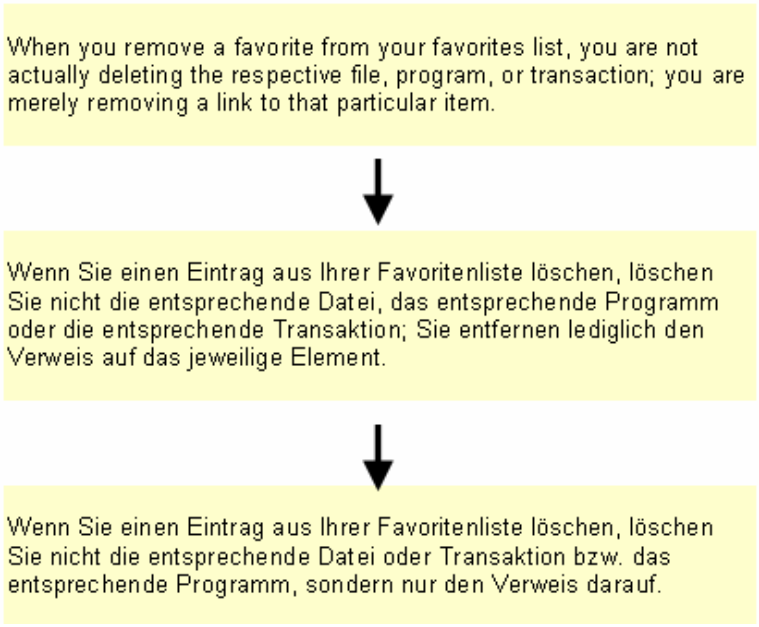


Figure 6: Example for a corrected text length.

Fig. 6 is split into three parts. The first part shows a piece of the original text from an English manual. The second part shows the direct translation to German. The third part is the corrected version of the German text. (In this case the length of the corrected version is very similar but not completely identical with the length of the English version.)

Forces:

- Your material contains text boxes that changed their length for some reason, e.g. because of changing a companies name, changing the terminology or translating content.
- The new length of the text boxes does not comply with the length required for those text boxes, e.g. if the text has to fit into a speech bubble.
- To change the size of the text box it is not enough to resize it e.g. by changing the font size. You must change its length.

Solution: If you want to correct the length of a text box you have to change the content. There are two possibilities regarding the length: Your new text is too long or too short. Texts, that are too long, have to be shortened and texts, that are too short, have to be lengthened.

To shorten a text you can try to find abbreviations. (Caution: Do not use too many abbreviations. This has a negative impact on readability and understandability.) In addition you can try to find synonyms that are shorter (e.g. "to make longer" – "lengthen"). If you use synonyms you must be careful that you do not change the meaning. You can also try to find phrasings that are shorter (see Fig. 6).

To lengthen a text you have the same possibilities: You can write out all abbreviations, you can try to find longer synonyms and you can try to find longer phrasings.

Known uses: This pattern is based on our own experiences as well as on the experiences of several other people working with text, like translators.

Consequences:

Positive:

- The changed text now has the correct length.

Negative:

- Depending on how you changed the length the quality of the text might have decreased.
- The text might have a slightly different meaning.
- If you use many abbreviations the readability of the corrected text might be not as good as it was before.

Related patterns: Not known

Used Patterns: None

7 Acknowledgements

We would like to thank Fernando Lyardet and Markus Schumacher for their valuable comments on this work. Special thanks go to James Coplien for shepherding this paper.

This work is supported by the German Federal Ministry of Economics and Technology in the context of the project Content Sharing.

The authors would like to thank SAP AG - SAP Research CEC Darmstadt, as well as KOM Multimedia Communications Lab at Technische Universität Darmstadt for supporting this work.

Contact Information:

Birgit Zimmermann[1, 2], Christoph Rensing[2], Ralf Steinmetz[2]

[1] SAP AG
SAP Research CEC Darmstadt
Bleichstr. 8, 64283 Darmstadt, Germany
birgit.zimmermann@sap.com

[2] KOM Multimedia Communications Lab
Technische Universität Darmstadt
Merckstr. 25, 64283 Darmstadt, Germany
{birgit.zimmermann, christoph.rensing, ralf.steinmetz}@kom.tu-darmstadt.de

For further questions and remarks please contact Birgit Zimmermann.

References

[1] Graham, I.: A Pattern Language for Web Usability. Addison Wesley, 2003

[2] Lyardet, F., Rossi, G.: Web Usability Patterns. In: Proceeding of EuroPLoP 2001

[3] Vogel, O., Zdun, U: Content Conversion and Generation on the Web: A Pattern Language, Published on EuroPLoP 2002

[4] Zimmermann, B., Bergsträßer, S., Rensing, C., Steinmetz, R.: A Requirements Analysis of Adaptations of Re-Usable (E-Learning) Content. Published in: Proceedings of EdMedia 2006

“Not just another conference”

Pattern language for conducting a successful niche conference

Cecilia Haskins
chaskins25@yahoo.com

Abstract: Niche conferences are designed to appeal to a specialized audience. They are crafted, not mass produced, by volunteers with a vision and the commitment to create an assembly for learning and networking. The quality of the conference is determined by all participants, not only the featured presenters. Such events are generally small in numbers, which means the budgets are tighter than those of larger conferences. This pattern language provides guidance to the committee of a niche conference to help them create a forum for the exchange of new ideas and “not just another conference.”

The context: a niche conference

The Merriam-Webster online dictionary contains the following definition of niche;

> 2 a: a place, employment, status, or activity for which a person or thing is best fitted … d: a specialized market.

This pattern language has been written to recommend techniques and solutions that are especially suited to organizing committees with the following attributes:

- Volunteers – not paid in any currency other than appreciation and the chance to attend the conference at no charge;
- Small committee size – ten or fewer volunteers, which in turn limits the ambitions set for the event, i.e., the conference is not designed to attract over 500 people or last more than three days;
- Professionals – which means that the primary expertise of members is in the field of the conference and no one in the committee is an expert at running events;
- Small budgets – which means there is very little money to spend for extras.

Due to the small number of participants and relatively intimate environment, a niche conference shares some of the characteristics of a writers’ workshop, especially the need to create an atmosphere of trust and goodwill. The author believes that the pattern language presented here is best set in the context of niche conferences. Of course, organizing committees for larger, well-funded events are welcome to use the good ideas embedded in these patterns.

History and background of this language

Historically, this pattern language derives from seven years of experience creating the ROOTS conference. The author is the conference visionary and had disjointed experience creating events before launching ROOTS. This pattern language is written primarily as a tutorial for new committee members and other volunteers in the Norwegian Computer Society (DND) who create similar events.

The story of ROOTS begins in Oslo in 1998 when the author attended a one day event of brief lectures followed by a panel debate between the distinguished speakers on the subject of OO techniques. Based on the strong attendance, the vision to create a longer event was conceived. A committee was formed from volunteer members of the DND with an interest in OO

technologies shortly thereafter and the first conference, a two day event, occurred in Bergen during the spring 2000 with a social connection to a local music festival, called at that time the Blues and Roots Festival.

The committee members held a retrospective after the first ROOTS conference. Jim Coplien helped the committee articulate a purpose for the conference resulting in the current formula with tracks for both technical and leadership participants. Martin Fowler gave us many helpful tips on what worked best from his perspective as a speaker, and advised the committee not to grow the conference into the thousands of participants, as was the then-current plan. As a result, ROOTS' invited speakers have shaped much of the evolving vision that motivates this conference. The planning committee concluded that the number of participants and the variety of themes should remain small. This was an important decision because with the general topic of "object orientation" there is a great temptation to incorporate diverse themes.

The committee confirmed its original intention of creating a forum for discussion and debate. They updated the concept to include both managers and practitioners – to attract groups of people who work together and need to create effective teams. In effect, the target audience mirrors the skills and interests of the committee members.

A successful conference exhibits the characteristic of meeting the expectations of both the committee and the participants. The ROOTS conference structure has evolved significantly based upon the feedback from attendees, who have placed great emphasis on maximizing interaction time for participants and speakers. For this reason pauses and lunches are scheduled with generous time allocations. Each year the program is designed, with the help of prior years' feedback, to attract this very unique audience. Even when an organization is only able to send one or two team members, we are aware that the learning is taken back and shared, much as bees returning to a hive.

Details about conference planning processes and methods are not a focus of this pattern language. Readers are encouraged to develop their own checklists and team structures. The essence of the ROOTS committee year-long planning process is summarized as follows. The organizing committee begins by creating a wish list of persons they would want to hear during the conference. From this, and feedback from prior years, the theme of the conference is determined. Individual members of the committee volunteer to take the role of personal host for a speaker. The speakers are asked to suggest what presentations or tutorials they would most like to deliver consistent with the theme.

Speakers are encouraged to be approachable to the small audience during lunch and conference social events. A unique gift is presented to each speaker as thanks and a memento of the occasion.

Every project depends on some critical success factors. They are listed here with commentary regarding how each factor is addressed by the ROOTS organizing committee.

a. Person in charge – as visionary, the author served as chair in the first year and again in subsequent years. As with all roles, the role of chair has rotated among the committee members. This has ensured that the vision has been internalized by all the volunteers who are involved in creating the annual event, that responsibility is shared, and that contributions are made in a risk-free and unstressed environment.

b. Proximity to community – every member of the organizing committee is a person who would otherwise attend the conference. This ensures that the themes, speakers and topics selected are of interest to potential attendees.

c. Planning and structure of the event – the conference structure evolved from a very successful event and has continued to evolve primarily based on feedback from attendees. Otherwise, the event is relatively unstructured (long pauses, BOFs), allowing participants to create a more personalized experience.

d. Individual sessions – once the speaker list is defined and confirmed, the speakers themselves determine the content of the conference by their suggestions for talks and workshops. By getting the right mix of speakers, and with very little direction, the overall program falls into place.

e. Standard components versus standard process – as indicated in item d. above, the process is standardized from year to year, but the content (or components) are variable based on the contributions of the participants, both speakers and attendees.

f. Cost control – the committee has great autonomy and no stress to make a profit, although great losses are not sustainable. Once a price per participant has been determined, the committee monitors registration and adds or subtracts optional items right up to the final weeks.

What is day-to-day life like during the construction of the conference? – The ROOTS organizing committee has grown into a tight-knit community that shares a common goal and has fun together. Meetings are infused with the sort of humour one finds among siblings and good friends. Every opportunity for collective meals or events is taken. When a problem arises, each member looks seriously at what contributions they can make for resolution without the attitude "that's not my concern." The committee currently exhibits many of the documented attributes of effective teams. Some of the members have been involved since the first event; at least one new member is added each year as other members rotate out of the committee. This keeps the viewpoints new, and helps resist stagnation.

The positive feedback from other conference organizers regarding this language suggests that there is a need for this knowledge. The author hopes that future users of the patterns in this language will also contribute to its improvement. Kindly send any comments and suggestions to chaskins25@yahoo.com.

The first pattern for this pattern language and the introduction of the language itself were presented during VikingPLoP 2002. Additional patterns have emerged and received generous workshop feedback during VikingPLoP 2003-2006, and PloP 2005. The intent of this author is that the language will mature, gradually include more original patterns and will eventually build on precedent patterns from other sources. The final result should be a collection of the essential elements that are required by conference organizing committees. The patlets that follow indicate the last event at which the pattern was included in a workshop. The entire pattern language is published for the first time in the proceedings of VikingPLoP 2006. The patterns can also be found on the ROOTS conference website, http://roots.dnd.no.

Summary of the Language

The patterns in this language are organized into four sections entitled Vision and Process, Atmosphere, Roles, and Customs. The patterns in the Vision and Process section identify the importance of identifying the core values of a conference and the process for creating the conference. The other pattern areas serve to reinforce this core.

Vision and Process

The ROOTS conference vision is to provide a forum for the discussion of new ideas and trends in OO development and to foster a spirit of team learning – TOGETHER.

1.	MARKETPLACE OF NEW IDEAS	presents the use of panel format	P04
2.	NEWS FROM THE FRONT	explains the use of experience reports	P04
3.	HONEY TO THE HIVE	program contains something to take back to the office and use tomorrow; a primary goal of the event	VP04
4.	COMMITTEE ARE EXPERTS	importance of diversity in the program	VP04
5.	PROGRAM MIRRORS PROCESS	how the conference program evolves and reflects the committee organization and processes	VP06

Atmosphere

Establish a setting for the conference that is comfortable, intimate and conducive to formal and informal dialogues. The technical aspects of the physical facilities should not distract the participants from their primary objectives; learning and networking.

11.	COFFEE ALL DAY LONG	importance of hospitality overall	VP05
12.	COMFORTABLE BEDS	importance of facilities, both hotel and conference	VP05
13.	WELCOME GATHERING	importance of becoming acquainted	VP05
14.	NOISY BREAK TIME	the importance of having enough time to begin and end conversations and the opportunity for professional friendships to develop during an event	VP04
15.	SAFE ENVIRONMENT	the importance of mutual respect in the exchange of ideas	VP04

Roles

Leadership and collaboration are important to achieve a successful conference.

21.	MOTIVATED COMMITTEE	excellent committees make excellent events	P04
22.	COURAGEOUS VOLUNTEERS	take the burden off committee during event	VP06
23.	LEADER GUIDES THE COMMITTEE	importance of leadership in the organizing committee	VP04
24.	STRONG SUPPORTING STAFF	importance of the facilities professionals	P04
25.	DISTINGUISHED SPEAKERS	importance of the invited speakers	VP05
26.	SPEAKER HOST	the importance of nurturing invited speakers	VP05
27.	ENGAGED PARTICIPANTS	importance of keeping participants involved	VP05
28.	PRICELESS SPONSORS	importance of financial support	VP05

Customs

Repetition of certain formulas re-enforces the comfort level for returning participants and speakers and creates a format for "continuing" dialogues between and within each conference event.

31.	FEEDBACK FORM	importance of asking for and using feedback	VP05
32.	ORDERLY REGISTRATION	Importance of first impressions	VP06

33.	RETURN OF PARTICIPANTS	the importance of having some of the same attendees returning for an annual conference	VP04
34.	SOMETHING TO REMEMBER US BY	gift selection for speakers, lottery	VP06
35.	ENVIRONMENTALLY CONSCIOUS CHOICES	paper free and recycling	VP06
36.	YEAR-ROUND WEBSITE	creating a source for continuous communication and reference material from the current and past conferences	VP06
37.	UNOFFICIAL GUIDE TO GET AROUND	create an informal connection	Plan
38.	PERSONAL VOICE	importance of the tone in communications	Plan
39.	NIGHT ON THE TOWN	social event related to location	Plan

References

James O. Coplien, "A Pattern Language for Writers' Workshops", Pattern Languages of Program Design, N. Harrison, B. Foote, H. Rohnert (Ed.), Addison-Wesley, vol. 4, 2000, pp 557-580.

Merriam Webster Dictionary Online, http://www.merriamwebster.com/dictionary/niche, last accessed 3.dec.2006.

Oscar Nierstrasz, "Identify the Champion", Pattern Languages of Program Design, N. Harrison, B. Foote, H. Rohnert (Ed.), Addison-Wesley, vol. 4, 2000, pp 539-556. http://iamwww.unibe.ch/~oscar/Champion/, last accessed 3.dec.2006.

Kai A. Olsen, "The Economics of International Conferences," **Computer,** June 2004, pp. 90-92.

Linda Rising, How to chair ChiliPLoP: a checklist for when to do what, 2004.

Acknowledgements

Sincere thanks to Richard Gabriel for inspiring the author to improve the work and for his many excellent suggestions for maturing the language. The author is most grateful to the numerous workshop colleagues for the gift of their time and insights.

1 – MARKETPLACE OF NEW IDEAS

…Ideally, an international conference provides a meeting place where the participants can present and discuss new ideas, research topics and results.[1]

❖ ❖ ❖

Conferences tend to encourage safe topics that are well established in the literature but these can be perceived as boring and having little value to a conference attendee.

Professionals with busy schedules are the target audience for most niche conferences. They hope to keep abreast and ahead of their industry by attending conferences because they do not have the time to read the journals where new research and practices are initially reported.

Conference organizers like to choose tightly constrained themes and encourage speakers to focus on presentations with recognizable content but they forget that conference attendees need inspiration as well as training. If participants are forewarned that they will not find presentation material in any book, they will feel honored to be a part of the process.

Both speakers and audience are challenged to be open and a spirit of trust builds between them. The conference becomes a SAFE ENVIRONMENT with enough interaction between DISTINGUISHED SPEAKERS and ENGAGED PARTICIPANTS that the new ideas are the subject of discussions. Speakers appreciate the opportunity to receive immediate feedback on how their messages are received and understood. In this manner, participants may explore possible application of the new ideas to their own workplace and the speakers have the benefit of refining both their ideas and their presentations methods based on the information received from the attendees. The organizers must ensure that pauses are long enough for conversations to start and continue.

Since a niche conference relies on the RETURN OF PARTICIPANTS, these conversations are an important factor to entice participants to return annually to share their experiences formally and informally and continue the cycle of sharing new ideas. In this marketplace ideas are

exchanged, not wares, and both merchant (speaker) and shopper (participant) receive something of value and are enriched by the experience.

Therefore:

Create an element in the program that is a forum for the discussion of new ideas. Invite highly reputable experts and encourage them to share new material with the audience that describes what they are trying to do in their current work.

The ROOTS conference begins every year with a first day format that encourages speakers to give brief insights into the latest trends they observe in the OO industry. The format includes a moderated panel which gives the speakers a chance to present counter viewpoints and to interact with the participants.

Select keynote speakers who stretch the boundaries of comfortable topics; plenary speakers need to be strong enough to invigorate and challenge the audience.

An atmosphere of trust is critical to the success of this pattern – see SAFE ENVIRONMENT. The final result is a NOISY BREAK TIME and highly ENGAGED PARTICIPANTS who return annually – see RETURN OF PARTICIPANTS. Returning participants may offer to present an experience report – see NEWS FROM THE FRONT.

Acknowledgements: http://www.haroldrigaud.net/works/marketplace.jpg
Marketplace, 20"x24", Original Oil on canvas

[1] Olsen, Kai A., "The Economics of International Conferences," *Computer,* June 2004.

2 – NEWS FROM THE FRONT

… a balanced conference program needs a mix of program elements.

❖ ❖ ❖

Conference attendees do not fully believe everything they hear from the podium, especially if the speaker is a well-know author or consultant with something to sell. What elements can be introduced into the program to establish credibility?

When creating a technical program, the planning committee needs to consider how best to help the participants integrate new material from the speakers and reinforce the learning experience. Famous speakers create a barrier to learning because their objectivity is questioned by the sceptical members of the audience.

Ironically, participants want to hear from top authors and consultants while at the same time doubting that the message is something that applies to their work situation. The information from famous speakers can be tainted by the fact that they have books and services to sell. Even when the best speakers include many real world examples in their presentations, they may fail to convince their audience of the full value of their message.

After listening to presentations from DISTINGUISHED SPEAKERS participants are ready to think about how they should use what they learned. Listening to a peer describe how they have used similar techniques to solve their own problems helps participants integrate new material from the conference. Concrete examples of how to apply a technique reinforce learning. The fact that the experience is shared by a person the participant can regard as a peer lowers the threshold of scepticism and resistance to new ideas. Hence, the conference becomes a more effective learning environment.

Therefore:

Balance the program between famous and non-famous people. The latter, as peers of the participants, present results from the work they are doing in experience reports.

A report delivered by a credible speaker who holds a peer status with their audience often carries more value and can be perceived as validating information received from sources that are more famous. Other participants are perceived as being more credible to the extent that they are not selling anything and merely sharing experiences to which the audience can relate.

The source of the experience may be industry, research, or academic, but the speaker is invited by the committee and the resulting presentation is not a formal peer-reviewed product.

❖ ❖ ❖

This pattern joins others in reinforcing the importance of communications among the participants as proposed in NOISY BREAK TIME. When possible, experience reports should reinforce the conference theme and demonstrate a concrete use of material presented by other speakers. Just as news reported by a person co-located with the news event has more credibility, so do experience reports constitute the conference equivalent of news from the front.

This pattern is related to patterns about how to find, invite and schedule experience reports. For more details see the Pattern Language *Identify the Champion* by Oscar Nierstrasz, 1999.

Acknowledgement: Victor Vasnetsov. News from the Front. 1878. Oil on canvas. The Tretyakov Gallery, Moscow, Russia.

3 – BEES TO FLOWERS; HONEY TO THE HIVE

... A conference must attract participants to succeed.

❖ ❖ ❖

What is the best way to attract participants to a conference?

Organizational training budgets generally support employee attendance at conferences and other training events. Often a team wishes to have multiple sources of new ideas and training and this in turn motivates the individual team members to attend diverse events. This single attendee approach often means that the information that is returned to the organization is missing the perspective of more than one viewpoint.

Tutorials and workshops generally contain high learning content. Organizing committees should vary the mix of topics such that everyone in the targeted audience has a good chance to find a subject that they want to learn more about. The evidence that a program has succeeded in this goal is when participants complain that they wanted to be in more than one place at the same time. When a person from a an organization wishes they could be in more than one program track, they often return in following years with additional colleagues to ensure coverage of the program material. Organizations that send more than one participant should experience an advantage of meeting multiple learning objectives and coordinated feedback on the themes of the conference.

From the economic perspective, the committee can consider offering a small incentive in the form of a price rebate for multiple attendees from the same organization.

Most speakers make slide presentations that contain good information but do not include the classroom interactions and concrete examples that make attendance desirable. If the presentation material is published, participants can see what they missed and make a justification for attendance by more than one member of the team in future years. Upon returning to their workplace, attendees can use these materials to explain the main learning points to their colleagues.

Therefore:

Create a program with coordinated concrete learning elements such that every participant takes away something that they can use on their next day on the job.

Good word of mouth is an important element to create annual participation for a niche conference. Organizers should create a program that encourages participants to return with colleagues in following years. Using the analogy of a beehive, sending more than one bee into a field of flowers yields the sweetest honey.

❖ ❖ ❖

Provide ample opportunities for the participants to meet and learn from others at the conference – NOISY BREAK TIME. Ensure that the DISTINGUISHED SPEAKERS are comfortable and willing to spend quality time with each other and the participants – WELCOME GATHERING.

It is important to the success of this approach that the tutorial and workshop leaders are respected persons and give good concrete and real-world examples in their material – DISTINGUISHED SPEAKERS. Reduce the frustration for the participants who miss a tutorial by providing the slides from all tutorials and workshops when permitted – YEAR-ROUND WEBSITE.

Reinforce the training elements with experience reports that address the themes – NEWS FROM THE FRONT. The program should contain a diversity of topics that are relevant to the needs of the workplaces of the targeted audience – COMMITTEE ARE EXPERTS. This pattern should motivate RETURN OF PARTICIPANTS.

Other conferences use their website to provide "arguments" that can be used to "convince the boss" to approve participation. See the following URL for examples:
http://srivaths.blogspot.com/2005/06/reasons-to-attend-nofluffjuststuff.html
http://www.jaoo.dk/convince/
http://www.spaconference.org/spa2005/convinceyourboss.html

Graphics acknowledgement:
http://www.puzzlesbyrussells.co.nz/images/bee.hive.with.bees.n_small.jpg

4 – COMMITTEE ARE EXPERTS

> *An expert is someone who has succeeded in making decisions and judgements simpler through knowing what to pay attention to and what to ignore.* (Edward de Bono)

... When planning a conference, the committee needs to integrate the goals and ideas set out in MARKETPLACE OF NEW IDEAS, NEWS FROM THE FRONT, BEES TO FLOWERS; HONEY TO THE HIVE, and FEEDBACK FORM.

❖ ❖ ❖

What guidelines should a committee use to determine the theme, content and format of the conference?

Conferences, by design, are limited in the time available during which a program must offer enough diversity to address the theme and keep the participants engaged. Despite an advertised theme, the audience has a diverse background and each participant has an individual learning objective.

To add to the challenge, each participant has a preferred method for learning. A variety of options regarding program format and content are available to an organizing committee:

- Plenary sessions with a speaker on stage behind a podium, such as keynote speeches
- Speakers on a dais interacting with audience, such as a panel
- Tutorials and workshops with a small, engaged audience
- Talks with practical vs. theoretical content
- Learn by doing vs. learn by listening
- Networking and discussions with others
- Conference banquets that offer an opportunity for participants and speakers to mingle
- Exhibitor area with vendors describing the newest products and tools.

Most conferences incorporate one or more of the above-mentioned program formats into their events. Those participants who prefer to listen and reflect may gain most from the formats with speakers. Panels are a good format in which to observe speakers interacting with each other. Question and answer periods should follow every speaker session and be integrated into tutorials and workshops. Exposing participants to a variety of methods for presenting information around a theme is also part of the learning experience.

When building the program, the committee should remember that there are at least two types of presenters. One is someone in the field who is an expert on a topic. In this case, the participants may agree or disagree with the speaker, and this can generate questions and discussions. The other type of speaker is someone from outside the field who brings in new material with which the audience may not be familiar. Presentations of the new viewpoints can stimulate discussions between participants for the period of the conference. These are very different experiences: but both are effective in engaging the participants.

Program variety also involves staying abreast of innovations in the field and offering sessions that introduce new topics without sacrificing sessions that provide updates on established technologies.

It is boring to sit for hours just listening – even to excellent speakers. Scheduled breaks provide an opportunity during which participants can give immediate feedback to speakers and validate their own knowledge by talking about new material. A conference attracts attendees with varying expertise, from novices to experts. A schedule that offers participants a generous amount of time in which to converse increases their opportunities to learn from each other.

Therefore:

Rely on the expertise and diversity in the committee to build the program using content and format options that appeal to each member. Committee members should encourage each other to exercise personal creativity when proposing speakers and themes for the conference.

When participants are presented with variety in the program, each element is a fresh experience and anticipation is high for the upcoming events. Structure the event so that participants who are too busy to attend an entire conference are able to choose the one day that is best suited to their personal objectives and learning style.

❖ ❖ ❖

Every year the organizing committee should considers adding new variety to the program. Use feedback from participants – FEEDBACK FORM – for suggestions about topics that interest them, ways to relieve stress on the schedule, their desire for longer or shorter pauses and the quality of experience reports – NEWS FROM THE FRONT.

Since the committee are also in the role of SPEAKER HOST, they can approach their DISTINGUISHED SPEAKERS as peers when negotiating the content of the presentations. It is a matter of expert judgement how much new material versus updates to prior themes should appear on the program.

Positive feedback from participants reinforces the committee decisions and supports this pattern. If the program keeps abreast of changing technologies, this gives additional justification to the participants for returning to the same event in subsequent years – RETURNING PARTICIPANTS.

Signs of a successful program are a NOISY BREAK TIME and ENGAGED PARTICIPANTS.

5 – PROGRAM MIRRORS PROCESS

... the content of a conference is determined by internal and external influences.

❖ ❖ ❖

There are many options for creating a project organization for planning a conference. The question is whether there is an optimal way to structure the committee to achieve the desired results.

Most teams have a leader, but beyond that, few roles are fixed in project planning committees. This means that communications within the group are essential to create a successful event.

A successful conference is highly dependent on STRONG SUPPORTING STAFF, the DISTINGUISHED SPEAKERS selected for the program, and the PRICELESS SPONSORS. It is desirable to establish a single voice for each external interface to avoid miscommunications, cross-communications and general confusion. Feedback and new ideas from these sources are important to the committee to create an annual event that is interesting every year.

Therefore:

Establish a committee organization that optimises communications channels by mirroring the desired content and format of the conference.

Committee roles imply a process (set of activities). Accordingly, the communication channels associated with each role will define the content of the conference. Establish committee roles that mirror the variety of internal and external interactions necessary to achieve the desired results. For example, if the conference program includes a conference dinner, one committee member should have sole responsibility for interfacing between the dinner venue and the person on the committee managing the attendance.

Here are some examples of roles and responsibilities:

- Leader – this role is akin to a conductor of an orchestra; often the face of the conference with high visibility – see LEADER GUIDES THE COMMITTEE
- Speaker Host – this pivotal role ensures smooth interactions with speakers; speakers determine the ultimate content of the conference by their selection of topic and the quality of their delivery – this makes each event unique – see SPEAKER HOST

- Secretary – keeps track of decisions taken and incomplete actions; liaison with the event sponsor; generates documentation that helps jump-start subsequent planning cycles
- Treasurer / Sponsorship – keep track of money in and money out; create and maintain a budget against which decisions about spending can be made; manage sponsorship interactions, both payments and the delivery of benefits – see PRICELESS SPONSORS
- Experience reports – this role manages the selection of and benefits accrued to reporters, which differs from other invited speakers
- Webmaster – maintains the YEAR-ROUND WEBSITE; responsible for technical aspects of the site and underlying registration database; the entire committee assumes responsibility for providing content
- Publicity / marketing – writing content for the website; taking contact with local press; generating materials for special constituencies
- Coordinator of volunteers and registration – this role maintains a procedure for registration and other conference duties; recruits students from nearby university or college – see COURAGEOUS VOLUNTEERS and ORDERLY REGISTRATION
- Coordinator of dinner / facilities – single point of contact for the conference venue and any off-site conference program elements, such as a NIGHT OUT ON THE TOWN
- Former committee members – people will leave the committee; however, they can provide insight, perspective and good advice without accepting other responsibilities; they usually provide a sponsor or other benefit that justifies attendance as a committee member
- Master of ceremonies – this role is usually filled by the chair, but should be filled by a person comfortable speaking in public; develop a script for each plenary session to avoid rambling.

Each of these roles has a unique contact with the service providers, participants, speakers, sponsors, and student volunteers. The nature of this interaction determines the tasks of the role and the process appropriate to achieve the desired results. The content of the program and the atmosphere of the conference are determined by how well each committee member performs their duties.

When planning an annual event, one goal is to keep the conference fresh and alive. This implies a close interaction with the DISTINGUISHED SPEAKERS, who provide innovative topics; with the STRONG SUPPORTING STAFF, who can suggest new menus or seating arrangements in the session rooms; with PRICELESS SPONSORS, who can provide a fun give-away item as a souvenir to participants, or an expert speaker. Each committee member is responsible to bring these new ideas into the committee to stimulate the creation of the event.

❖ ❖ ❖

Many of these roles are the subject of individual patterns (as indicated above). It should be noted that depending on the range of duties and the capacity of individual volunteer committee members, one person may fill multiple roles. This is also desirable to avoid a bloated committee or persons attending at no cost without having made any contribution. Likewise, some roles, such as publicity, may receive contributions from multiple members. **MOTIVATED COMMITTEE** contains additional information about the internal interactions within the committee.

11 – COFFEE ALL DAY LONG

Make an ordinary gathering a special event by including food. [Do Food[1]]

… the organizing committee wants the quality of the atmosphere to match the quality of the technical program.

❖ ❖ ❖

The organizing committee are not event-hosting experts but must establish criteria for selection of the conference facility.

Most committee members have attended at least one conference before volunteering to serve on a conference organizing committee. This experience makes each member qualified to hold an opinion about facilities and service during an event. But with a limited budget, it may be necessary to temper wishes with the reality of available venues.

Nearly everyone can agree that rooms should be an ample size to hold the expected audience, chairs should be comfortable and audio-visual equipment should be adequate to accommodate the presentations. Experience suggests that hospitality factors are the quickest way to create a satisfying conference environment. Delegates who are hungry or thirsty will lose interest in the proceedings in spite of their intentions to remain engaged.

In many cultures the availability of a well-brewed cup of coffee (or tea) represents hospitality. Even if economy constrains other features, just this gesture can create a warm and welcoming atmosphere and convey the committee's concern for the comfort of the participants.

Therefore:

Select a facility that excels in providing a comfortable and hospitable environment.

An example of a hospitality criterion is to choose a location that serves coffee all day long. The committee should not overlook availability of AV services and appropriately sized meeting rooms. A visit to the location during business hours should help confirm that the ambient temperature of the site is maintained at a comfortable level.

❖ ❖ ❖

Coffee is, of course, a metaphor for tea, soda, water and coffee that is readily accessible, and provided at no additional cost to the participant. Refreshments should be balanced and include healthy alternatives to cakes and pastries. Advertising and registration should be clear about the inclusion of such amenities. Vending machines should be considered only if no other alternative exists.

Most conference centers will quote the daily rate to include pauses and lunch. This pattern suggests that the committee should negotiate to have liquid refreshment available to participants throughout the day for a reasonable cost.

As an example, PloP brings in a local vendor who specializes in boutique coffees and teas on the first day of workshops. This pattern is consistent with “Do Food” from Fearless Change, quoted in the heading.

Providing something to eat and drink during breaks, as well as during lunch, reinforces the social nature of these pauses and contributes to the success of **NOISY BREAK TIME** and **ENGAGED PARTICIPANTS**.

[1] *Fearless Change: patterns for introducing new ideas*. Mary Lynn Manns and Linda Rising. Boston MA: Addison-Wesley, 2005.

12 – COMFORTABLE BEDS

> *"Some industry experts say the trend [to provide superior bedding] stems from a new generation of travellers who expect hotel amenities to equal or exceed the comfort of home. *"*

… the organizing committee strive to be good hosts to their invited guests and conference participants.

❖ ❖ ❖

How does the organizing committee create a 24-hour positive experience for participants during the conference?

This pattern deals with the hospitality factors surrounding the conference and is closely related to COFFEE ALL DAY LONG. After a long day travelling or attending sessions, participants need to relax and some may need to check with the office. When on the road, most travellers still expect the comforts of home; security, a clean, quiet room, a good shower, and a comfortable bed.

In major cities, the cost of hotel accommodations can exceed the conference registration. Travellers also appreciate locations that do not involve lengthy bus or taxi rides from the airport. Punctuality and attendance are both enhanced if a single facility includes both quality conference rooms and quality overnight accommodations.

Therefore:

Select a conference facility that cooperates with a reputable hotel. Since overnight expenses are a part of the total cost package for the conference participants, keep these as low as possible without sacrificing comfort.

One way in which a small conference can keep the overall budget low and reduce the total cost for the participants is to host the conference in a smaller city with a nearby international airport. Many cities are investing in such locations as a way to boost the local economy. Another workable approach is to use university or research conference facilities – although the room standards will not be as high as hotel standards.

A hotel that provides rooms that meet or exceed modern standards will ensure that the overall conference experience is not degraded by a bad night's sleep. Today this standard includes extra features such as wireless internet connections. It is a bonus if the hotel includes breakfast in the room price, is within walking distance of the conference facility, and is willing to provide a conference discount.

❖ ❖ ❖

When accommodations are connected to the conference facility, the committee is able to achieve benefits from cooperating with the STRONG SUPPORTING STAFF, who will help ensure that conference guests are pampered. Ensure that the hotel is aware which guests are the DISTINGUISHED SPEAKERS and assigns them rooms that are away from street noises and other distractions.

Collocating DISTINGUISHED SPEAKERS in the same hotel further increases their opportunities to interact with each other and with participants. A hotel with a comfortable hotel bar is an added bonus as this provides an excellent setting to include late arrivals for the WELCOME GATHERING and increases the opportunities for casual encounters between speakers and other participants, enhancing NOISY BREAK TIMES and ENGAGED PARTICIPANTS.

[*] “Hotels focus on a 'good night's sleep,'” Yvonne Teems. *Cincinnati Business Courier,* October 21, 2005.

13 – WELCOME GATHERING

"Welcome to Bergen" gathering of the "extended committee" in private home the evening before the start of ROOTS. (ROOTS archive, 2003)

… the conference is about to begin; the DISTINGUISHED SPEAKERS arrive.

❖ ❖ ❖ ❖ ❖

After months of planning, the committee and speakers are finally meeting, sometimes for the first time.

International speakers are accustomed to travel and hotel rooms – these are a part of the normal fabric of their lives. Often they meet people for the first time and go immediately to work despite the fact that they can arrive jet-lagged and unfamiliar with a new town. Often they do not know anyone in the new city, not even other speakers. When this is not the case, they are still hesitant to intrude into the private time of fellow travellers. In either case, they appreciate a distraction to keep them awake but without the stress of performing for a crowd.

After months of planning, the committee is ready to enjoy the company of their invited speakers and play the role of good hosts who make their guests comfortable. A small informal meeting between hosts and speakers is part of the non-monetary reward for all their efforts.

During the event the speakers and MOTIVATED COMMITTEE must function as an "extended committee" providing value to the participants.

Therefore:

Schedule an optional informal gathering to welcome the speakers on the evening of their arrival before the conference begins.

By joining an informal gathering, the DISTINGUISHED SPEAKERS do not have to find a meal, navigate a strange city, or eat alone (unless they so choose).

❖ ❖ ❖

The gathering serves to break the ice for persons who have not yet met, but will spend the better part of 2-3 days together. Those already acquainted often pick up unfinished conversations from their last meeting, and the mood is generally light and jovial. The welcome can take place in the home of a committee member, or a nearby restaurant. One criterion is that the location should encourage a congenial, informal atmosphere.

14 – NOISY BREAK TIME

... it follows from MARKETPLACE OF NEW IDEAS and PROGRAM VARIETY that conference attendees need time in which to discuss new ideas with each other and the speakers.

❖ ❖ ❖

How should a conference capitalize on the richness of the technical program?

The human mind has a limited absorption period. Scheduling pauses is necessary for the comfort of the participants and for their enjoyment of the event – see COFFEE ALL DAY LONG.

Many planners schedule very short pauses in the hope that participants will return quickly to the program sessions but this often backfires and the participants do not return as hoped, preferring to stand in the hallways to finish a conversation. Longer pauses are more likely to ensure that people return to the sessions, even if this means fewer sessions.

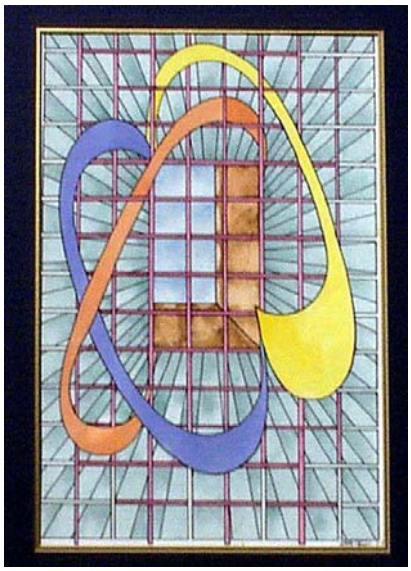
Therefore:

Schedule long pauses in the program to maximize the amount of time available for conversations between participants and speakers and among participants. Pauses should be longer than 15 minutes, especially if there are snacks served.

❖ ❖ ❖

Maximizing the amount of time available for conversations between participants and speakers and among participants creates a congenial professional atmosphere. Acquaintances may grow into professional collaboration outside of the event, such as forming study groups and planning other extra-conference gatherings. Many participants return each year as much for the educational content as for the chance to commune with valued colleagues. The conference begins to develop a personality formed by the returning participants – see RETURN OF PARTICIPANTS.

15 – SAFE ENVIRONMENT



... as suggested by MARKETPLACE OF NEW IDEAS, conferences are more interesting when presenters offer new ideas rather than repeating well rehearsed material.

❖ ❖ ❖

How do you encourage speakers to share their unpolished ideas without feeling vulnerable to criticism, while at the same time encouraging audience feedback?

Authors survive by seeing their work published and widely read and applied. For works published without a critical review process, they receive their first feedback from the market they aim to influence. This may provide a rude awakening for the author. Clearly, authors benefit from the feedback of both peers and colleagues. This is one of the many reasons they attend conferences.

Authors also need a forum where they can feel comfortable presenting their suspicions and findings for the first time in public. When they do, they need to place their presentation into a context of "this works" or "this may work better, but I'm not sure yet." This may create an impression of uncertainty, which can make both participants and speakers uncomfortable.

Generally, a group of people develop trust through shared experiences over time. If a conference is to generate a community of trust, the participants must be together for a longer period of time than is usually afforded by a single event. However, a session with new material is best scheduled early in a conference. This is the point at which everyone is most vulnerable resulting in polite reactions and unbiased viewpoints.

In a conference setting, an informal session early in the program, advertised as a MARKETPLACE OF NEW IDEAS, is a refreshing change from formal training sessions and well rehearsed tutorials. The speakers are willing to release unpolished work in this setting, knowing that everyone has been alerted to the fact that these are new ideas. This session must be combined with long breaks during which the material is the subject of face to face discussions. Attendees feel less uncomfortable asking questions about new material in a

personal conversation than in a plenary room. In a non-threatening, casual conversation, presenters are able to understand how their material was received and understood.

Therefore:

Provide a SAFE ENVIRONMENT where an author/presenter can receive useful feedback directed at their material rather than at themselves.

Modifications to the material can be made later to integrate the feedback. Material from this session is generally not published until the author refines the work and offers it for wider publication.

❖ ❖ ❖

This pattern is a precondition for MARKETPLACE OF NEW IDEAS. The ROOTS conference committee begins to build trust by assigning a SPEAKER HOST to help the DISTINGUISHED SPEAKER prepare his or her thoughts before the conference. The hosting process provides the speaker with a preview of the conference themes and gives the committee a chance to offer feedback if a proposed topic feels too risky. In addition, the conference has enjoyed a high rate of returning participants every year since the first event. If the participants return annually, the conference attendees have the opportunity to develop into a strong community of trust – see RETURN OF PARTICIPANTS.

Encourage use of the FEEDBACK FORM as a mechanism to provide comments that are difficult to give face-to-face.

Acknowledgement:
A Safe Place, Transparent Watercolor, Gunther Bach, www.netgallerybach.com/

Related Patterns:
"Safe Setting" and "A Community of Trust" from *A Pattern Language for Writers' Workshops*, James O. Coplien, 1997.

"Safe Place" from *Knowledge Hydrant*, Joshua Kerievsky, 1999.

21 – MOTIVATED COMMITTEE

Planning committee for rOOts 2002, picture taken by T. Fossnes

Many niche conferences are planned by volunteers of sponsoring organizations, such as computer societies. Such volunteers are not paid in monetary currencies and often execute planning activities alongside a full-time job.

❖ ❖ ❖

A niche conference planning committee requires the concerted efforts of all members to achieve their desired results. Conference planning spans up to a year, which is a long time for volunteers to remain interested and motivated.

Volunteers have many priorities in their everyday lives such as family, work and personal time. They must be strongly motivated to participate in activities that occur outside working hours.

When a committee forms, the members should take the time to know each other and agree on their mutual objectives. One of the first tasks is to evaluate realistically the amount of work to be done and to identify the interests and abilities of each individual committee member. If there are too few volunteers a committee can quickly burn out.

Each committee member has the option to commit to the activities, results and level of effort s/he can contribute. This includes the option to be a backup support for another member. It is important that each member feels that s/he is making a meaningful contribution to the end result and that the efforts are appreciated by the entire committee. Since each member selects their own tasks no one is pressured to do work they find uninteresting.

Over a long planning period volunteers may become demoralized and drop out. Individuals are initially motivated to remain active as they see the steady progression of early results, for example confirmed speakers and the forming of a draft program. During the later and more hectic phases, they are motivated by the opportunity to participate in an event that they created.

Every member accepts one or more roles that define the nature of their commitments. The role of committee chair is especially critical (see **LEADER GUIDES THE COMMITTEE**).

A committee that works together over extended periods of time should deliberately encourage members to accept new roles and learn new skills with the support of the experienced members. This enhances motivation by avoiding a boredom factor. **SPEAKER HOSTS** tend to continue to host the same person if they are re-invited since a personal relationship is critical to that role.

As individuals accept responsibility and meet their commitments trust grows in a committee. This trust leads the members to feel a strong personal obligation to deliver the promised results. Most volunteers accept responsibilities in good faith. A group of people can usually determine if one member is taking on too much responsibility.

Failure of a committee member to deliver promised results may jeopardize the conference. The schedule should always contain enough slack such that, in the event of unexpected personal obligations, another member can step in to complete a task without creating undue stress. When an unanticipated challenge appears, such as the cancellation by a speaker, other members step in and make suggestions that help to resolve the issue.

The literature is filled with cautionary advice about team building and project management. A conference planning committee is generally modelled after a self-selected team. It is also helpful if the members have personalities that blend well in group settings. The committee engages in honest and open communications to prevent avoidable failures. A healthy, motivated committee exhibits the characteristics of a well-functioning team such as mutual respect, a common language from shared experience and they smile a lot when together.

Therefore:

A successful conference is created by a motivated committee.

This simple pattern is a critical success factor for a successful niche conference. The quality of the committee often determines the quality of the conference and the atmosphere of the event mirrors the working environment of the committee.

❖ ❖ ❖

Suggestions for motivation and team building techniques are available in the project management and organizational literature. Well defined and executed roles – see LEADER GUIDES THE COMMITTEE – are important planning tools. The opportunity to work personally with a DISTINGUISHED SPEAKER may serve as a motivation for some committee members – see SPEAKER HOST. Removing some of the stress related to facilities and hospitality also contribute to the success of this pattern – see STRONG SUPPORTING STAFF.

Related Pattern:
Cope/Harrison, "Self-selecting Team," in Organizational Patterns of Agile Software Development, Prentice Hall, NJ: Upper Saddle River, 2004, p. 124-125.

22 – COURAGEOUS VOLUNTEERS

ROOTS volunteers, 2004

... after working for an entire year, the organizing committee is ready to enjoy the conference.

❖ ❖ ❖

A great deal of work is required during a conference to ensure a seamless experience for the participants.

There is a long list of hidden activities that underlie every event. Registration must proceed smoothly, audio-visual equipment is expected to work perfectly, and laws regarding maximum persons in a room must be observed. The committee is certainly qualified to handle these details, but they create a distraction from the enjoyment of participation.

Therefore:

Recruit volunteers from a local university or college to do the "heavy lifting" during the conference.

Student volunteers receive the benefit of attending the conference for no cost, and should also receive some gift from the committee. In return, they perform a number of duties. They staff the registration desk each day of the event, and must learn the procedures related to an ORDERLY REGISTRATION. At least one volunteer is present in every session of the conference – which means there must be as many volunteers as there are concurrent tracks. They must learn to use the audio-visual equipment, and coordinate computers that are swapped between speakers in the plenary session. They are expected to monitor attendance in the sessions and to ensure that there are not more persons present than legally permitted (usually a fire safety limit). In the event of overflow, they are expected to check the name tags and prioritise those who remain according to the declared registration selections – this might take a bit of courage.

❖ ❖ ❖

A side benefit is that after this experience the vision has been imparted and some volunteers join the MOTIVATED COMMITTEE with a good background in the conference.

23 – LEADER GUIDES THE COMMITTEE

Leadership is the ability to guide individuals and organizations to optimum levels of performance and service. [anon]
Leadership is practiced not so much in words as in attitude and in actions. [Harold Geneen, Chairman, ITT Corp.]

... When an all volunteer conference planning committee forms, one of the first questions is "who is in charge?" Often the leader is determined by a political process that does not take into account the abilities of the individual. Small committees are especially vulnerable.

❖ ❖ ❖

A small organizing committee cannot afford dissention or poor group dynamics during their conference planning cycle.

When volunteers come together for after-hours activities, there is a temptation to leave the practices of their workplace in the workplace. As a result, otherwise well-organized people often flounder when trying to complete relatively simple tasks, such as inviting a speaker or writing a press release. Individuals who are models of good team spirit on the job often can become protective of their volunteer tasks and fail to share critical information with the group. This may be the result of not taking the volunteer activity as seriously as a job, or the fact that most committee efforts are for the most part a lonely activity and it is easy to forget the rest of the group when they are not around.

The leader should encourage volunteers to accept activities that can be well defined and have known need dates and interdependencies. Such activities are frequently the result of brainstorming within the group, which serves as a good technique for forming a team. In this way each committee member is sensitized from the beginning to the cooperative nature of their assignments. The leader also accepts assignments and serves as a source of good example to the other volunteers by the way in which they fulfil their commitments.

When a committee is small, there are fewer people to pick up an extra task if an activity is overdue or not being done properly. One of the primary duties of the leader is to provide encouragement and facilitate maximum cooperation between committee members. The leader watches the schedule and gently reminds individual volunteers when a task is becoming due and offers help if it is needed. The committee leader should also "externally" observe group dynamics and use team building techniques. This latter responsibility is handled by arranging for the team to have fun together as well as work together, for example by sharing a meal and celebrating success.

The committee leader respects the value of the time of each volunteer and establishes an agenda and set of objectives for every meeting. It is best if another team member accepts the responsibility to record decisions and open action items in minutes from each meeting. It is also advisable to separate the role of financial manager (treasurer) such that it is the prime responsibility of one team member to watch the economic situation and advise the committee on related decisions, such as registration pricing and rebates. A leader who attempts to write

minutes and manage the budget while leading the team will soon burn out unless the role is a full-time occupation.

It is also helpful if the leader knows something about project management or is willing to delegate these duties to a deputy. In the case of a political appointment, the titular leader is more of a figurehead. A figurehead may not have the background skills for managing a project or group of people. When there are more experienced persons on the committee they may resent the leader and not feel appreciated for the contributions they are making or are able to make. Also, without the mantle of leadership, a committee member working behind the scenes may be viewed as disruptive or bossy, and thereby be ineffective in helping the team achieve their goals. In this instance it is highly advisable to create a two-tiered leadership structure with the figurehead on top buffered from the actual volunteers by a layer of working leaders whose good example serves as an inspiration.

Therefore:

Choose a committee leader capable of treating the conference planning activity as a project and instituting appropriate team building and project management techniques.

Just as the conference is a reflection of the committee, so the committee is a reflection of its leadership. A facilitating leader has good people skills and is able to bring out the best efforts of each volunteer.

❖ ❖ ❖

This pattern offers additional insight into the critical roles within the MOTIVATED COMMITTEE. The enthusiasm and good example set by the leader is critical to the success of the committee. The guidance of prior chairpersons who have been through the experience also provides additional support and helps maintain morale in the committee.

> *Many things we do as conference chair one would say could be done by a professional planner, or by the facility itself. While true, this costs money, and we have enough trouble with attendance without charging twice as much to reduce the chair's workload. Think of it as a big party you are planning, with a few more logistical issues to address. (Thanks to John Letourneau, How to chair ChiliPLoP)*

Related Pattern:
Don Sherwood Olsen and Carol L. Stimmel, "Guiding Hand" from *The Manager Pool*, Addison-Wesley, 2002, pp 3-4.

24 - STRONG SUPORTTING STAFF

In 2002, three days before the ROOTS conference start date, a nationwide strike was imposed on the facility. On short notice, the facility manager arranged for alternate accommodations for all delegates, appropriate facilities for two of the conference's three day format, and provided discounted bus transportation to bring delegates downtown for the conference banquet. This left the committee to deal with rearranging one day of sessions. The committee was able to meet all the program obligations. Since most delegates knew of the strike, they were very tolerant and had a good time anyway.

…the conference planning committee has created the technical program and needs a place to hold the event.

❖ ❖ ❖

The organizing committee has minimum influence on the non-programmatic elements of a conference, but these factors have a large influence on the overall experience for participants. The committee must rely on contractual arrangements for services to anticipate all contingencies.

The physical comfort of the conference facilities and the quality of meals strongly influence a participant's overall impression of a conference. A poor sound system, too cold or too warm ambient temperature, uncomfortable seating or a skimpy lunch can undermine an excellent technical program. Committee volunteers are generally experts in their professional domain and able to mange the technical program. The same volunteers are novices in the field of delivering the hospitality services related to a conference and have relatively little control over the environmental elements of the conference. For this they must rely heavily on the facilities contractor to create a comfortable physical environment and provide an adequate level of services for the participants. Cost is always a factor when balancing what is affordable against what is desired.

The hospitality staff that services a convention or congress center understands the many details that are important to their guests. A professional staff provides services in a timely and friendly manner and is available to address the small disruptions that inevitably occur. Members of the planning committee are happy to reduce their workload wherever feasible, and are busy enough networking, being hosts and attending to the technical program. If the facilities manager and staff can be trusted to provide a comfortable environment for the participants, this reduces stress for everyone.

Over time the planning committee and facilities staff develop a collaborative style and this cooperation increases their effectiveness during the conference. Loyalty is an effective way to create a collaborative cooperation between the committee and facilities provider. This may have the added benefit of favorable cost quotations and a priority assignment when requesting dates for future events. Loyalty creates a virtuous cycle in that the committee is pleased with results and wants to return, and the facility manager is able to count on return business and works hard to earn the continued patronage.

Therefore:

Select a facility provider that shares a compatible viewpoint with the organizing committee and include the facilities manager as a member of the committee in the role of strong supporting staff.

As a de facto committee member, the facilities manager should receive updates regarding all elements of the program that could impact the ability of the locale to support the conference – for example, total number of attendees, number and size of rooms needed, starting and ending times each day, AV and special equipment needs and the schedule for lunch and coffee breaks.

The organizers of the rOOts conference have been privileged to be supported since 2000 by the staff of the Bergen Congress Center (BKS) in Bergen, Norway. Over the years, the congress center has expanded their concept for coffee breaks from the initial format of coffee and tea available all day, to including a light snack in the morning or afternoon, and finally to serving a small buffet of cheese in the morning and cakes and fruit in the afternoon. BKS has also expanded lunch from a fixed portion meal to an open buffet. All of these changes occurred without the urging of the committee but were completely consistent with the level of service the committee wanted to provide and were offered with nominal cost increases.

In 2001 the rOOts conference needed rooms for concurrent tracks and had to provide their own AV equipment because of poor foresight. The next year BKS anticipated the need for AV as soon as the number of rooms were requested and provided a day-round AV specialist to handle any questions. In the following years, the level of AV support has kept pace with technological innovations, including building-wide wireless Internet services.

As illustrated by the opening story, a number of exceptional situations have arisen throughout the years that have reinforced the relationship between the committee and BKS.

An effective committee – MOTIVATED COMMITTEE – is made more effective by close coordination with the facilities provider. Since the relationship is personal, continuity in points-of-contact is recommended, as discussed in PROCESS MIRRORS PROGRAM.

The facilities provider is an important factor in creating a comfortable atmosphere for the participants – see COMFORTABLE BEDS and COFFEE ALL DAY LONG. The employees of the hosting facility become a part of the team – treated with respect and appreciation. As an example, PloP conference organizers traditionally give Swiss Chocolate gifts to the staff at Atherton.

25 – DISTINGUISHED SPEAKERS

Distinguished speakers conversing during a break at ROOTS 2004 (from left, Trygve Reenskaug, Alistair Cockburn and Michael Jackson). (ROOTS archives)

… a small, niche conference tries to differentiate their event and offer extraordinary value to participants.

❖ ❖ ❖ ❖ ❖

How can an organizing committee create an interesting program that will attract both sponsorship and participation?

Sponsorship and good attendance are dependent on an interesting and quality program. Participants are inevitably drawn to conferences at which published authors or renowned speakers are giving presentations and tutorials.

Sponsorship is required to keep an event affordable, and sponsors also prefer to be associated with a quality event filled with well-known speakers and good content.

Speakers of international calibre can command high fees and they often have hectic schedules. At the same time speakers have an altruistic side that motivates them to help a worthwhile event by reducing or deferring their normal honorarium. They appreciate a relaxed environment and the opportunity to meet and talk with each other. They also appreciate the opportunity to receive feedback and interact with conference participants if they are not stressed or overwhelmed.

An important feature of a niche conference is its small audience. The expectation on the part of all participants is that this intimacy will provide opportunities for increased interaction between the presenters and the participants. Speakers are expected to be present throughout the event to maximize these opportunities.

Therefore:

Invite the most distinguished speaker in the field. At the same time, offer the enticement of interacting with other best-in-field distinguished speakers.

A small conference achieves its primary goals by creating a program filled with highly qualified presenters. The invitation should make clear the limited funding of the event, and request a reduction or deferral of honorarium. It should also be understood that the speaker is expected to be present for all sessions, unless other arrangements are discussed.

❖ ❖ ❖

Simplify and personalize the interactions with each speaker by assigning a SPEAKER HOST. The presence of a spouse should be encouraged and the spouse included in all meals and social activities. These include pre- and post-conference gatherings, such as the WELCOME GATHERING.

Cover all travel expenses and any other reasonable expenses in the conference budget. The hosting committee should arrange hotel reservations – see COMFORTABLE BEDS – for the speakers such that all billing comes directly to the organizers and the speakers are spared the inconvenience of paying this bill and applying for reimbursement.

There is always the possibility that a speaker will not accept the committee invitation. There have been experiences where ROOTS was scheduled against a "bigger" event and lost potential speakers to that event. The end result is that for every speaker who says "no thanks" there are two that say "yes." And often those who decline in one year because of a conflict in schedule make it clear they would like to be invited again and participate in a following year.

Another consequence to watch for is the overbooked speaker who experiences a sudden change of plans. When the committee creates a program, some local persons should be identified to substitute in case of a late cancellation. Sometimes your speakers are distinguished in more than one field, and are willing to step in to fill an empty session. In either event, sickness or cancellation is always a risk when planning an event.

The one caveat when selecting the speakers is to collect early feedback on the speaking qualifications of the invitee. A committee can avoid disillusioned and disappointed participants if they pre-screen the list of invited speakers by asking people who have heard them give at least one presentation. Some well-known authors are not as talented in face-to-face or audience circumstances.

Related Pattern:

Mary Lynn Manns and Linda Rising, "BIG JOLT" from *Fearless Change*, MA: Boston, Prentice Hall, 2004, pp. 107-109.

26 – SPEAKER HOST

"Speaker with host" Alan J. O'Callaghan (to right) with his host Jon Stefan Fraçzak in a role-playing exercise during ROOTS 2003. (ROOTS archives)

… the organizing committee needs to attract DISTINGUISHED SPEAKERS.

❖ ❖ ❖ ❖ ❖

Speakers of international calibre are in great demand, how can a small conference attract them?

Speakers are first and foremost human. Even if they achieve fame as guru and author, they value positive experiences and having a good time. They like to take holidays and meet new people.

Speakers consider conferences to be highly impersonal. Smaller conferences can offer a degree of intimacy for invited speakers that allow them to meet with their colleagues and with the participants of the event. Participants can also find conferences to be impersonal and feel very privileged when they can spend some quality time with a presenter.

If speakers are satisfied with the organization of a conference they are more willing to consider repeating the experience. The relationships established from prior years work to the advantage of both the speakers and organizing committee.

Therefore:

Assign each speaker a host who is responsible for creating a personal connection with them.

As discussed in MOTIVATED COMMITTEE, the planners have many reasons to remain active. One of these may be the opportunity to establish a personal relationship with one or more of the speakers by accepting the role as SPEAKER HOST. During the many communications, speakers and host often develop friendships that transcend the conference interactions.

It is important in the first invitation to establish the fact that this conference is special and will offer a more personalized experience. The host is responsible for all communications with a speaker before, during and after the conference. This eliminates confusion for the speakers regarding where to address questions and where to send materials. The behaviour of hosts creates the atmosphere of a SAFE ENVIRONMENT for the speakers. This is important in the discussions with the speaker about the length, format and content of their presentations.

When a speaker arrives in town, every effort is made for the host (or another member of the committee) to meet them. This is especially important if the speaker does not speak the local language. During the event, the host is available to assist in any matters that arise while the speaker is in town. Usually, the host will enquire on the availability of a speaker for the following year to kick-off the ongoing negotiations to bring back appreciated and versatile presenters.

❖ ❖ ❖

Occasionally a committee member becomes ill (or a father for the first time) and other members of the planning committee will step in temporarily to provide continuity in the hosting. Sometimes the chemistry is better between different individuals, in which case, experience shows, the host and speaker assignments may shift from year to year – this is normal and should not be a problem within the MOTIVATED COMMITTEE.

27 – ENGAGED PARTICIPANTS

> *"Conferences are fun and they should be, but to get the most for your money, you need to remember the reason you are there. Your priority is to do the best job possible representing your organization. Conference schedules can be killers; try to avoid conference burnout. Pace yourself — Get enough sleep — Take advantage of scheduled free time to relax."*[1]

… the planning committee have DISTINGUISHED SPEAKERS and PRICELESS SPONSORS, now they need registered participants.

How can the organizing committee help participants combat conference malaise?

Planning the program is only the half the job of a conference organizing committee. The other half is concerned with delivering value in the total conference experience. If participants are faced with the choice of sitting and listening to speakers all day, or taking stroll through the downtown shopping area, or jumping on the internet to read mail, the option with the highest enjoyment and personal payback will usually win.

With a limited budget, smaller conferences can not afford the entertainment elements found in larger events. And if participants are idle in this day of blogging, they take the time to share their feelings with the world.

Therefore:

Create interactive elements in the program that keep the participants interested and engaged in the conference.

The key to warding off outside distractions is to keep a lively pace and avoid situations that generate the feeling of fatigue. Short speaker presentations sprinkled with frequent and generous pauses lets participants stretch their legs and interact with others. Scheduling a question and answer period after every presentation gives participants ample time to formally interact with the speakers and keeps listeners alert and intellectually engaged.

Exhibitors or book stalls are another way to provide a change of scenery within the conference and away from the lure of the city. A closing lottery (drawing) with fun prizes, such as t-shirts, free registrations or autographed books from the presenters, encourages the participants to stay for the closing keynote plenary session.

Informal Birds-of-a-Feather gatherings are also effective to break the ice and get everyone talking to each other. For years the PloP workshops have had regularly scheduled games orchestrated by games-master George Platt to facilitate introductions, creativity and just fun. *"George is a ``tangential thinking co-ordinator." During the conference he will play some games with us. These should provide a relaxed atmosphere that encourages open communication and discussion, as well as keep you fit and clear-minded for all the sessions."*[2]

Attention is drawn to the patterns NOISY BREAK TIME, COFFEE ALL DAY LONG and COMFORTABLE BEDS. These address the importance of hospitality, comfortable facilities, effective media aids, ample refreshment and time built into the program to facilitate interactions.

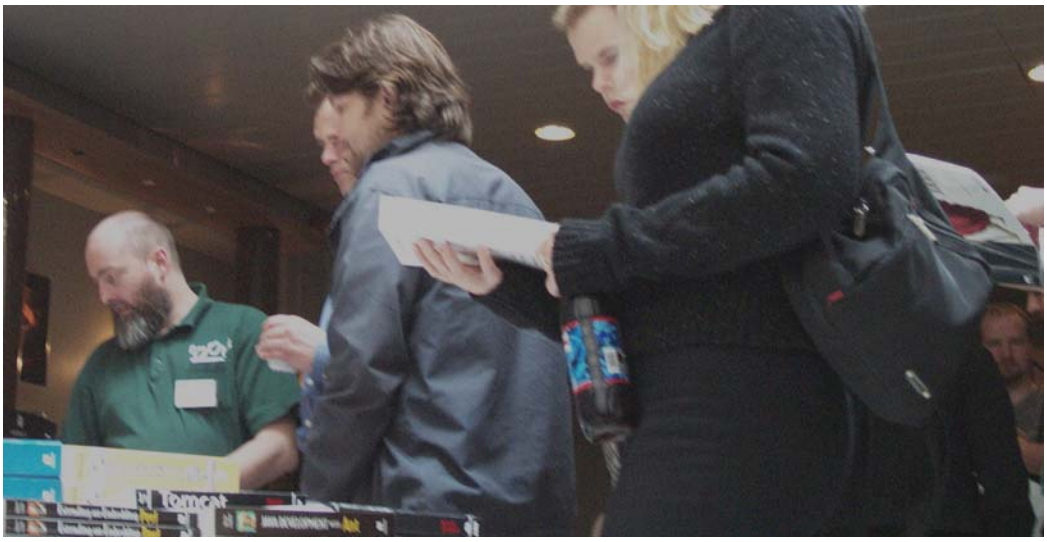
There is always the possibility that something could go awry. In such situations the diligence of the committee members is the very important. The committee should sense when people are uncomfortable or unable to hear a presentation. Listening to the participants is the best way to keep them engaged. As Linda Rising suggested, it is a good sign if no one is checking their email or writing a blog. See the following URL for advice given to delegates about how to attend a conference - http://faculty.ed.uiuc.edu/j-levin/conference-hints.html.

Acknowledgements:

[1] http://www.uwec.edu/dc/AP/complexthoughts/ConferenceAttendance.html

[2] http://www.cs.wustl.edu/~schmidt/PLoP-96/events.html, and ... /europlop-96/misc.html

28 – PRICELESS SPONSORS



Discounted books from the publisher attract attention during ROOTS 2003. (ROOTS archive)

… the organizers of a niche conference must put a price on registration.

❖ ❖ ❖

How can the organizers of a niche conference keep the registration affordable?

The planners of smaller conferences must calculate the break-even very realistically. By its definition a niche conference does not attempt to appeal to a large audience. If the registration fees are kept low to attract maximum participation, there is not much room in the budget for extravagance or to buffer against things that might go wrong.

For most participants the ability to attend a conference is directly related to an affordable price. The decision to attend an event is a complex value computation that seeks to minimize the cost of travel and conference fees while maximizing conference content and contact with DISTINGUISHED SPEAKERS.

Sponsorship is an alternative way to generate funding for a conference. Securing sponsors becomes a secondary activity for all members of the planning committee. Alternative funding is a critical success factor for the event to break even.

Therefore:

Use the financial support of sponsors to keep the event affordable. Offer value to sponsors in exchange for their priceless support.

One way to attract sponsors is to create a quality conference that translates into prestige for those affiliated with the event (for example, by association with DISTINGUISHED SPEAKERS). The sponsorship program should be designed to give visibility to sponsors and to show appreciation for their support.

Niche conferences focus on technical themes. The participants expect all of the sessions to be non-commercial. This means that they are able to decide for themselves when and how to be exposed to the message of sponsors, which means they are less resistant to commercial information when they receive it. This is the ideal communication environment for the sponsors. Hot links from the YEAR-ROUND WEBSITE are one of the most visible and long-lasting benefits for sponsors.

Sponsorship is an exercise in creativity. Money is not the only commodity of value to the organizers. A publisher willing to offer book discounts, a small consultancy willing to donate conference bags or a richer sponsor willing to pay a premium for a special conference rate all contribute to the success of the event. A sponsorship program that allows recognition for levels of support means that a sponsor with limited resources can also participate as a sponsor and feel appreciated.

Examples of sponsorship alternatives to cash payments are: provide speaker gifts; provide something of value for a lottery (drawing); provide workers for the registration table; provide a conference bag or t-shirt; provide equipment; print the proceedings (or provide CDs); provide discounted products (books are most popular); send a large population to participate.

❖ ❖ ❖

Recognition is usually instantaneous with signing an agreement. The conference webmaster should publish the logo and establish the hot-link immediately. The committee treasurer should issue an invoice, and follow-up that the sponsorship funds have been collected before the end of the conference to avoid unpleasant collection negotiations that can cloud the planning for the next event.

31 – FEEDBACK FORM

(Photo http://www.uniquecanes.com/new/Main_Feedback.php)

… the conference is nearly over, next year's event is already taking shape.

❖ ❖ ❖

How can the organizing committee for an annual event learn what they have done correctly and where there is room for improvement?

After a long planning interval the event finally happens. Most of the committee are able to participate and should have a good idea about the reactions of others to the program, the speakers and the facility. But relying solely on the experience of the committee members is too biased to be helpful for planning subsequent events. The committee members need to hear from the other participants.

Asking the participants for their opinions is not guaranteed to solicit an honest or helpful response. Relying on verbal feedback risks misunderstanding or not capturing the opinions of every participant, including the speakers.

Therefore:

Insert a feedback form in every registration package, collect them before the end of the conference and use them when planning subsequent events.

There are many things that a planning committee would like to know about how the event was perceived by the participants. A feedback form should be designed to solicit this information during the conference, while reactions and ideas are still fresh. The number of questions asked needs to be balanced against the number of questions a respondent is willing to answer. The questions should address the facilities, the program and the registration process. An area should be provided for optional suggestions about future programs and suggestions for improvements. Feedback from satisfied (and unsatisfied) participants is one of the best ways to augment the creative ideas of the committee.

❖ ❖ ❖

If the participants are convinced that their feedback will be taken into consideration by the committee when planning future events, they will provide it gladly. It is important to remind participants that their feedback is important, and make it easy for them to return the form. A

fun technique to encourage returns is to offer the option of signing the feedback to be eligible for a drawing for prizes at the end of the conference, after the closing keynote speaker.

Having feedback in physical form supports analysis of past performance, and in negotiations with the facilities provider should improvements be required. Innovative ideas for future presentations and presenters make the job of the MOTIVATED COMMITTEE much easier the following year.

Use the YEAR-ROUND WEBSITE to solicit ideas not offered on the physical feedback form during the event. Indicate in the program when a theme or speaker is the result of using the feedback form. Encourage anonymity if it will yield more information and place a mailing address on the form for those who inadvertently bring it home, but want to return it later.

An alternative, and fun, way to collect feedback on site is to establish a conference timeline and conduct a mini-retrospective after the closing plenary. This option does require the services of an experienced facilitator, and may be too cumbersome to do every year.

32 – ORDERLY REGISTRATION

... first impressions are lasting impressions; the registration process is the first impression most participants receive of an event.

❖ ❖ ❖

Registration looks deceptively simple, but any delay quickly causes a queue to form. Arriving participants, eager to start their day, are not happy standing in line for their conference materials.

Registration is a necessary element of every conference. The process reassures the participant that they are welcome and expected. This may be the first acknowledgement that payment has been received. At a minimum, most events require a conference identification to authorize entry to the sessions, especially in venue that is shared with multiple events.

The typical registration package contains an identification tag and some materials for navigating the conference. These usually involve multiple items. Assembling these items on the day of the event demonstrates a lack of preparation and is discourteous to the waiting participant who is anxious to get a cup of coffee or network with colleagues before the first session.

Some participants, such as the speakers, should not be subjected to the standard process. They may be preoccupied or mentally preparing to speak and should be given special treatment.

Long queues and confusion in the registration process sets an expectation that the conference will be poorly managed and tedious.

Therefore:

Plan early for an orderly registration.

An orderly registration begins with the very first contact the participant has with the event. Seemingly unrelated factors, such as the ease of understanding of the program, can influence the process. As a technical note, a poorly defined or unreliable database creates stress for those preparing invoices, attendance lists, and name tags.

In the final days of preparation before the event, the committee and volunteers should have a joint meeting during which they prepare all materials that will be handled during registration.

These include a name tag (with holder), handouts from sponsors with the program, site map, proceedings CD, feedback form, and other give-aways, such as a conference bag. Arrange to receive these materials at least one week before this meeting.

It is also very important that the correct name tags and receipts are given to each individual, and that a record is maintained of who has arrived and registered. As people may not think to stagger their arrival, expect that there will be some queuing, but make provision to form 2-4 lines to handle peak times.

Persons staffing the registration desk should be trained and know the procedure without needing to ask unnecessary questions. The procedure should be as smooth as possible, with minimum verbal exchange, to shorten the time required with each participant. Staff should arrive at least 30 minutes before the first participants are expected during which they should set up the registration staging area and receive final instructions. A written checklist can be helpful if there are many details or exceptions.

The committee member in charge of registration should be nearby during registration to handle exceptions promptly. Other committee members should also loiter in the registration area to greet DISTINGUISHED SPEAKERS and PRICELESS SPONSORS. When practical, their materials should be maintained separately, and given to them by a committee member so that they can avoid any queue.

Most participants should be pre-registered, but develop a form to handle onsite registration, and customize some blank name tags for hand-written id. These blank tags also may be needed to handle the situation where someone forgets or looses their pre-printed id during the event.

Finally, arrange for the area to look attractive and be well marked so that participants can quickly determine where they should be, and feel welcome. Flowers and banners help achieve these goals. Provide adequate signage to reduce the number of queries for directions during the busy registration period.

❖ ❖ ❖

Orderly registration is closely linked to COURAGEOUS VOLUNTEERS and YEAR-ROUND WEBSITE. The former most often staff the registration area, and the latter is the mechanism by which registration information is received from the participants.

Some other items that are useful to have in the registration area are extra name id holders, tape or other affixer for posters, cough drops, pens, and markers. In preparation for equipment malfunctions or other emergencies, also have on hand blank CD-ROM or USB memory, laser pointer, and backup PC.

33 – RETURN OF PARTICIPANTS

... the committee has implemented **BEES TO THE FLOWERS; HONEY TO THE HIVE**, **SAFE ENVIRONMENT**, and **NOISY BREAK TIME**.

❖ ❖ ❖ ❖ ❖

How does a niche conference survive in the competitive market?

Marketing literature is filled with advice on the importance of keeping a customer and the relatively lower energy and cost threshold for selling to an existing customer over generating a new customer. The attendees of a conference are the customers and their repeat business is core to the survival of an event.

Software professionals are deluged with opportunities to attend conferences, vendor training and university sponsored courses. Since most professionals must also do real work, and the cost of these events is sometimes quite high, they must exercise discretion in selecting how to invest their time and money. The length and cost of an event or training session are usually important factors. But the return on investment measured in the value of the learning and the opportunity to build a professional network are also important.

Therefore:

Create an atmosphere and continuity in the program that entices participants to return annually.

One advantage of a small conference is that people have a much better chance to meet and become acquainted, especially if the pauses are long enough to facilitate conversations – see **NOISY BREAK TIME**. The opportunity to meet with professional colleagues and friends becomes additional motivation to return to an event.

One advantage of a niche conference is that the theme is focused and the audience is defined by professionals who have an interest in the area. This allows the organizing committee to create continuity in their planning by building on previous years' programs. This continuity helps participants justify their decision to attend and to invite colleagues and team mates when they return, especially if they can point to something valuable to the job that they

learned from the previous event – see **Bees to Flowers; Honey to the Hive**. Even if they are not able to persuade friends, their attitude translates into positive word of mouth advertising.

A positive experience creates a predisposition to attend an event annually. Make participants feel welcome from their first moments – see **Orderly Registration**. This pattern is reinforced during the conference by **Engaged Participants** and after the conference by maintaining a **Year-round website**.

When the number of participants is small, it is possible for the members of the organizing committee to actually get to know some of the participants. This creates a reciprocating relationship between planners and attendees who continue to return, like boomerangs.

❖ ❖ ❖ ❖ ❖

The planning committee must be rigorous in their use of the **Feedback Form** when planning each year's event to maintain continuity while adding innovative elements suggested by the participants themselves.

Here is an example of the desired effect the event wants to have on its participants;

> *Ahh EuroPloP. My most favouritest of conferences. I've been looking forwards to getting back there for years, and I really will make it in just a forthnight. ... the venue is great, the people wonderful, the drinks are strong. I could go on and on. Almost a perfect conference to attend: "whether you like food or sleep, or story-telling or singing, or just sitting and thinking best, or a pleasant mixture of them all." -- JamesNoble*

Photo acknowledgement: http://www.boomerangpassion.com/decouvrez/histoire.php

34 – SOMETHING TO REMEMBER US BY

... when an event is concluded, a souvenir reminds us of special people and places.

❖ ❖ ❖

The conference will create special memories, but how can the committee thank the speakers for their time and efforts in a small, but tangible way that keeps these memories alive?

Planning for an annual event includes planning for its continuity. Continuity is related to returning participants, both speakers and their audience. Paying participants have received a valuable package during registration including proceedings and other give-aways. However, as contributors to the knowledge materials, speakers probably will not use the proceedings very often in the time between events. Add to this the challenge that frequent conference-goers receive many items and are hard to please.

Therefore:

Select a tasteful souvenir that speakers can take away with them as a memento to remember the conference and the people they met.

Souvenirs are a matter of individual taste. Criteria for choosing an object are size, practicality, and the ability to be imprinted or embossed with the conference logo. Many speakers use only carry-on luggage, so small items are appreciated. Avoid items that have sharp points or lasers as these are confiscated during air travel, and would entail an extra mailing expense. Items that are fragile also may not survive the return home.

The ROOTS committee has given letter openers, laser pointers, key chains, coffee mugs, and umbrellas as speaker and committee gifts. As you can see, we learned about the lasers and sharp objects first hand! It is not possible to predict what will happen with the items, but things that can be put on a desk and are symbolic are recommended. Other items may be given to children or friends and not kept.

❖ ❖ ❖

This pattern is related to **DISTINGUISHED SPEAKERS**, **MOTIVATED COMMITTEE**, and **RETURNING PARTICIPANTS**.

35 – Environmentally Conscious Choices

... the best way to conserve resources and raw materials is not to use them in the first place.

❖ ❖ ❖

Many decisions that a conference planning committee makes influence cost and environmental impact.

Some actions are both expensive and make ineffective use of natural resources. The most wasteful use of raw materials is the extensive printing of brochures and other mailing pieces that fail to hit their target and become trash, with no guarantee of recycling. The cost of printing and mailing is often the largest single expense for an event. The second largest expense is printed proceedings, unless they are very small.

Name tag holders and compact discs are plastic, which is made from fossil-fuel sources.

Therefore:

When making decisions consider environmentally conscious options that are also cost effective.

A cost conscious committee can easily avoid expensive postage and printing costs by not mailing any promotional materials or printing conference proceedings. The only exceptions to this paperless policy are the program and feedback form. The former should be condensed, 2 pages to a side and front and back to form a portable program, easy to slip into a pocket. The latter is archived by the committee, which justifies printing; unless a mode for collecting online feedback is in place. Provide the proceedings via download from the YEAR-ROUND WEBSITE.

Continue in this practice by limiting the volume allocated to sponsors hand-outs in the registration package. The conference bag can be smaller, and the paperless message is reinforced. Consider cloth registration bags over plastic; they are just as easy to emboss with the conference logo.

Collect the plastic name holders at the end of the conference for reuse in the next event. At ROOTS, over 50% of the plastic is returned every year. A feedback question regarding a paperless conference receives overwhelming approval every year, with only 1-2 persons expressing a desire for more paper.

❖ ❖ ❖

This pattern contributes to maintaining the lowest cost possible, which is a key factor for the success of niche conferences. A paperless policy is possible by making responsible use of email to alert potential participants to updates in the **Year-round website**. The website also supports an eventual progress to eliminate distributing the proceedings on a CD-ROM (a fossil-based material).

36 – YEAR-ROUND WEBSITE

`http://roots.dnd.no`

... an annual event needs to keep potential participants interested and informed.

❖ ❖ ❖

Online registration is expected of a technology event. How can the cost of a temporary web presence be justified?

Organizing a successful annual event involves building a "brand recognition" that facilitates word-of-mouth advertising. Positioning the event as a source of new ideas and trends requires more than a once-a-year contact with the professional community. Media relations are also important, and journalists have their own schedule for collecting and reporting news.

The cost of a domain name and an internet server location are generally based on annual rates, so temporary usage is not cost effective. Likewise, inconsistencies in user interfaces creep into the implementation each time a site is re-constructed.

Therefore:

Maintain a year-round website, with a professional user interface and continuously updated content.

A permanent web presence allows prior and potential participants to visit the site whenever they wish. They can receive newsworthy updates about relevant events in the field, or download the materials from a prior conference. Maintaining an archive of the conference proceedings online ensures that professionals in the area will visit year-round to retrieve information when they need it. For example, in 2006, the ROOTS committee discontinued the practice of putting proceedings on a CD-ROM (fossil-based material) and only provide papers and session materials online.

People with questions can reach the committee at any time. Providing access to a forum that accepts contributions from visitors also stimulates year-round visitations to the site. Visitors who create a site registration login are also readily accessible for notifications about upcoming events from the committee.

PRICELESS SPONSORS also benefit from a year-round presence. Having their logo continuously on display demonstrates their support and commitment to a quality event.

eCommerce gurus stress the importance of continuously updating a website's content. This puts year-round pressure on the committee to keep providing updates or risk the consequence that the site will only be visited in a time-window surrounding the event.

Communication experts also stress the importance of the tone in communications, or the PERSONAL VOICE. This means the committee needs both a professional and competent webmaster and at least one member willing to serve as content editor for the site. The webmaster maintains a consistent style, consistent use of the logo, and fluid navigation.

Some of the participants will arrive from another city. They will rely on the website to keep them informed of travel and weather information. Providing an **Unofficial Guide To Get Around** can be used to create an informal connection with new visitors to the city.

❖ ❖ ❖

A conference committee using this pattern language has no paper contact with potential participants (**Environmentally Conscious Choices**); uses online collection to ensure an **Orderly Registration**; and relies heavily on **Return of Participants**. All of these patterns are supported by a year-round web presence.