Table of Contents

Welcome to PLoP 2008 ................................................................................................................................. 1
PLoP® 2008 Conference Proceedings ........................................................................................................... 2
PLoP 2008 Conference Description ............................................................................................................. 2
Invited Talks .................................................................................................................................................. 3
  "Learning & Teaching Design Patterns" ..................................................................................................... 3
  "Explaining and Exploring Design Patterns" ............................................................................................. 3
Other Workshops .......................................................................................................................................... 4
  SPaQu'08 ................................................................................................................................................... 4
  UI Patterns Workshop ............................................................................................................................... 4
Writer's Workshops ...................................................................................................................................... 5
  'Design & Architecture' group ................................................................................................................... 5
  'Software & People' Group ........................................................................................................................ 5
  'Processes and Services' Group ................................................................................................................ 6
  'Security & Quality' Group ........................................................................................................................ 6
Committees ................................................................................................................................................... 7
  Conference Organization Committees ...................................................................................................... 7
  Shepherding Committee ............................................................................................................................ 7
  Programming Committee .......................................................................................................................... 8
Welcome to PLoP 2008

Welcome to PLoP '08, the 15th Conference on Pattern Languages of Programs, a premier event for pattern authors and pattern enthusiasts to gather, discuss and learn more about patterns, pattern writing, pattern reviewing, shepherding, software development, collaboration, and more, much more.

To accomplish this, the conference program offers a rich set of activities that altogether promote a friendly and effective environment to share expertise, and to give and get feedback from fellow authors.

The pre-conference activities started Friday morning at the BootCamp, a special session aimed at people new to patterns and/or PLoP, led by Linda Rising and Robert Hanmer.

Writers' Workshops are the primary focus of our time at PLoP and it will be during them that we will discuss and review each other’s papers in a very fruitful way. We have four groups of six papers each, which were selected from an initial set of around 40 submissions, and after a considerable period of shepherding. Papers of the Writing Group will have in addition the opportunity of being evolved during PLoP with the mentoring of very experienced pattern writers. We are excited to have two Invited Talks which will be time to get inspired and energized by the words and thoughts of Joshua Kerievsky, and Rebecca Wirfs-Brock on hot topics related with design and learning. But there is more. Other activities, such as the 'Birds of a Feather' (BoF), or the Monday’s Workshops/Focus Groups let you informally organize your own session about topics you are interested in, or to attend already organized working sessions. Just announce them or subscribe to them!

After the conference, the papers are strongly encouraged to be further evolved in order to accommodate the suggestions for improvement gathered during the discussions at the conference. A final version of evolved papers will be published in the ACM Digital Library as PLoP 2008 Proceedings.

And last but not least, we have the Games, a well-established and very important activity at PLoP. Guided by Robert Hanmer, the games will help us all on ice-breaking, to exercise our body and mind, to collaborate better, and to reinforce a community of trust. Some of the games have become 'traditions', while others will be a surprise.

This year PLoP is not in the beautiful scenery of Allerton Park, the original PLoP location, where most conference editions took place. The notable exception is PLoP06, which was collocated with OOPSLA’06, in Portland, Oregon. This one is again co-located with OOPSLA, in this wonderful city of Nashville, Tennessee, the city of music.

We would like to thank all authors, shepherds, reviewers, and Program Committee members for their time and collaboration with PLoP. Thank you!

Ademar Aguiar and Joe Yoder, PLoP Chairs
Pattern Languages of Programs (PLoP®) conference is a premier event for pattern authors and pattern enthusiasts to gather, discuss and learn more about patterns and software development.

Preliminary versions of these papers were workshopped at Pattern Languages of Programming (PLoP) ’08 October 18th - 20th, 2008, Nashville, TN, USA. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. Copyright is held by the authors.

**PLoP 2008 Conference Description**

Joseph Yoder, General Chair  
The Refactory, Inc.  
joe@refactory.com  

Ademar Aguiar, Program Chair  
Universidade do Porto  
ademar.aguiar@fe.up.pt

Pattern Languages of Programs (PLoP) conference is a place for pattern authors to have their pattern languages reviewed by fellow authors. The purpose of PloP is to promote development of pattern languages on all aspects of software, including design and programming, software architecture, user interface design, domain modeling, and software process. Domain-specific patterns were encouraged for PLoP 2008.

PLoP 2008 was held in Nashville, TN, October 18-20 in conjunction with OOPSLA 2008.

**We invited contributions from practitioners and researchers on:**

- Patterns and pattern languages
- Critiques of patterns and pattern languages
- Research on patterns and pattern languages
- Case studies of the use of patterns and pattern languages

PLoP is different from other conferences. It is run in the "writers' workshop" style, as described in Richard Gabriel's book. Before the conference, authors interact with a "shepherd" who helps them improve their paper to make it as ready for PLoP as possible. A program committee reviews the papers for final acceptance after they have gone through the shepherding process. The writers workshops provide more feedback, and so authors revise their paper again after PLoP. The papers here are the version produced by authors after PLoP, not the ones reviewed at PLoP.
Invited Talks

At the PLoP '08 conference a number of invited speakers spoke on topics pertinent to pattern writing and developing trends in the software community. Following are list of the invited talks, including speaker information and a short synopsis of the talk:

"Learning & Teaching Design Patterns"

Joshua Kerievsky, Saturday, 19, 16:30-17:30 (Room Belmont)

Joshua Kerievsky has been programming professionally since 1987. He founded Industrial Logic, a company specializing in patterns, Extreme Programming (XP), and other techniques for more successful software development. He began his career as a professional programmer on Wall Street, where he developed numerous financial systems for credit, market and global risk departments. Kerievsky is an active member of the patterns and XP communities, and the author of many articles, simulations, and games.

Joshua recently published a book called RefactoringToPatterns.

"Explaining and Exploring Design Patterns"

Rebecca Wirfs-Brock, Sunday, 20, 09:00-10:00 (Room Belmont)

In 1989 Kent Beck and Ward Cunningham introduced CRC (Class- Responsibility-Collaborator) cards to the OOPSLA crowd as a tool for teaching object-oriented thinking. In that classic paper, they also hinted at the power of using CRC cards as a technique for gently, gradually introducing complex designs. This talk re-introduces several informal techniques that can be helpful in deciphering patterns as well illustrating new patterns. And just for balance, we'll briefly look at how UML (Unified Modeling Language) can be simply used to express design subtleties. There's a time and place for both informal and more formal views.

Rebecca Wirfs-Brock is an internationally recognized leader in the development of object design methodologies and is a consultant to enterprises of complex object architectures and designs. She invented the set of development practices known as Responsibility-Driven Design. Among her widely used innovations are use case conversations and object role stereotypes. Via her courses and conference tutorials she has taught object design concepts to thousands of programmers.

She is the regular design columnist for IEEE Software and the author of the classic text, Designing Object-Oriented Software. Her most recent book, Object Design: Roles, Responsibilities and Collaborations, was published in 2002. She also blogs regularly.
Other Workshops

SPaQu'08

The 2nd Workshop on Software Patterns and Quality (SPaQu'08) was held as a workshop at the 15th Conference on Pattern Languages of Programs (PLoP '08), to discuss the theoretical, social, technological and practical issues related to quality aspects of software patterns, including security aspects. The papers listed were presented at the SPaQu'08 workshop.

The proceedings include a "Report on the 2nd Workshop on Software Patterns and Quality (SPaQu'08)" written by Hironori Washizaki, Nobukazu Yoshioka, and Eduardo B. Fernandez.

The following are the accepted peer-reviewed papers that were accepted and presented at SPAQu. The report and three reviewed papers are included in the PLoP Proceedings.

"DEQUALITE: Building Design-based Software Quality Models"
Foutse Khomh and Yann-Gaël Guéhéneuc

"Quality of Test Specification by Application of Patterns”
by Justyna Zander-Nowicka and Pieter J. Mosterman

“Abstract security patterns”
by Eduardo B. Fernandez, Hironori Washizaki and Nobukazu Yoshioka

UI Patterns Workshop

UI Patterns workshop focused on how to recognize interface patterns, how to write them, how to organize a library of them, how to complement them with code and stencils, and how to design, prototype and build with them.

The workshop was conducted by Erin Malone, founder of the Yahoo! Design Pattern Library, Christian Crumlish, curator of the Yahoo! Design Pattern Library, and Lucas Pettinati, User Experience Lead for the Yahoo! User Interface Library.
## Writer's Workshops

### 'Design & Architecture' Group
**led by Ralph Johnson**

<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Patterns for Data and Metadata Evolution in Adaptive Object-Models&quot;</td>
<td>Hugo Sereno Ferreira, Filipe Figueiredo Correia, Leon Welicki</td>
</tr>
<tr>
<td>&quot;Freeway Patterns for SOA systems&quot;</td>
<td>Vinod Sarma, Srinivas Rao</td>
</tr>
<tr>
<td>&quot;Enterprise Architecture Management Patterns&quot;</td>
<td>Alexander M. Ernst</td>
</tr>
<tr>
<td>&quot;Patterns for Understanding Frameworks&quot;</td>
<td>Nuno Flores, Ademar Aguiar</td>
</tr>
<tr>
<td>&quot;The Dynamic Factory Pattern&quot;</td>
<td>Leon Welicki, Joseph W. Yoder, Rebecca Wirfs-Brock</td>
</tr>
<tr>
<td>&quot;A Pattern Language for Developing Analog to Digital Converter Data Sampling Firmware&quot;</td>
<td>Sachin Bammi, Peter Swinburne and Adefeyike Odutayo</td>
</tr>
</tbody>
</table>

### 'Software & People' Group
**led by Linda Rising and Joshua Kerievsky**

<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Learning and studying Interaction Design through Design Patterns&quot;</td>
<td>Miguel Carvalhais</td>
</tr>
<tr>
<td>&quot;Continuous Feedback Pedagogical Patterns&quot;</td>
<td>by Kathleen A. Larson, Frances P. Trees, D. Scott Weaver</td>
</tr>
<tr>
<td>&quot;Thoughts on Weak Links and Alexandrian Life in Scrum&quot;</td>
<td>Pam Rostal</td>
</tr>
<tr>
<td>&quot;Additional Patterns for Fearless Change&quot;</td>
<td>by Mary Lynn Manns, Linda Rising</td>
</tr>
<tr>
<td>&quot;The Relation between Design Patterns and Schema Theory&quot;</td>
<td>Christian Kohls, Katharina Scheiter</td>
</tr>
<tr>
<td>&quot;Fundamental Banking Patterns&quot;</td>
<td>Lubor Sesera</td>
</tr>
</tbody>
</table>
'Processes and Services' Group
led by Lise Hvatum and Bobby Woolf

"Runtime Mix'n Match Design Pattern"
Paul G. Austrem

"Deferred Cancellation. A Behavioral Pattern"
Philipp Bachmann

"Handling Transactional Business Services"
Geert Monsieur, Lotte De Rore, Monique Snoeck, Wilfried Lemahieu

"A Pattern for Monitoring Scenarios to Handle State Based Crosscutting Concerns"
Mark Mahoney, Tzilla Elrad

"Coordinator-Worker-Context Process Pattern"
John Liebenau

'Security & Quality' Group
led by Bob Hanmer and Brian Foote

"The Secure Blackboard Pattern"
Jorge L. Ortega-Arjona, Eduardo B. Fernandez

"A Catalogue of Bug Patterns for Exception Handling in Aspect-Oriented Programs"
Roberta Coelho, Awais Rashid, Uira Kulesza, Arndt von Staa, Carlos Lucena, James Noble

"Patterns for the Secure and Reliable Execution of Processes"
Eduardo B. Fernandez, David laRed Martinez

"Web Security Patterns for Analysis and Design"
Takao Okubo, Hidehiko Tanaka

"Patterns for ADT Optimisation"
David J. Pearce, James Noble
Committees

The PLoP Conference would not be a success without the volunteer help of the shepherds and program committee members. The shepherds devote hours of their time to helping authors improve their papers pre-conference. The program committee members help organize the conference, handle requests, and communicate with attendees.

We would like to thank all those who helped make PLoP 2008 a complete success.

Conference Organization Committees

<table>
<thead>
<tr>
<th>Role</th>
<th>Chair Name and Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conference Chair</td>
<td>Joseph Yoder (The Refactory Inc., USA)</td>
</tr>
<tr>
<td>Program Chair</td>
<td>Ademar Aguiar (FEUP &amp; INESC Porto, Universidade do Porto, Portugal)</td>
</tr>
<tr>
<td>Publicity &amp; BootCamp</td>
<td>Linda Rising (Independent Consultant, USA)</td>
</tr>
<tr>
<td></td>
<td>Bob Hanmer (Alcatel-Lucent, USA)</td>
</tr>
<tr>
<td>Publications</td>
<td>Pam Rostal</td>
</tr>
<tr>
<td>Registrations</td>
<td>Jason Frye (Hillside Group, USA)</td>
</tr>
<tr>
<td>Web Design</td>
<td>Ana Ferreira</td>
</tr>
<tr>
<td></td>
<td>Miguel Carvalhais (id:D / FBAUP, Portugal)</td>
</tr>
<tr>
<td></td>
<td>Jason Frye (Web Content Writer, USA)</td>
</tr>
</tbody>
</table>

Shepherding Committee

- Alejandra Garrido
- António Rito Silva
- Berna L. Massingill
- Bob Hanmer
- Cecilia Haskins
- Christian Kohls
- Daniel May
- Eduardo Fernandez
- Eugene Wallingford
- Fernando Castor Filho
- Hironori Washizaki
- Jorge L. Ortega Arjona
- Joseph Bergin
- Joseph Yoder
- Kyle Brown
- Linda Rising
- Lise Hvatum
- Marcelo d’Amorim
- Mary Lynn Manns
- Michael Weiss
- Neil Toussaint
- Ofra Homsky
- Paul Adamczyk
- Peter Sommerlad
- Roberta Coelho
- Rosana Teresinha Vaccare Braga
- Scott E. Schneider
- Terry Terunobu Fujino
- Uwe Zdun
- Wolfgang Herzner
Programming Committee

- Ademar Aguiar (INESC Porto/University of Porto, Portugal)
- Joseph Yoder (The Refactory Inc., USA)
- Linda Rising (Independent Consultant, USA)
- Richard P. Gabriel (IBM Research, USA)
- Bob Hanmer (Alcatel-Lucent, USA)
- Uwe Zdun (Vienna Technical University, Austria)
- Paulo Borba (Federal University of Pernambuco, Brazil)
- Eduardo Fernandez (Florida Atlantic University, USA)
- Rosana Teresinha Vaccare Braga (ICMC, University of São Paulo, Brazil)
- Hironori Washizaki (National Institute of Informatics, Japan)
- Peter Sommerlad (Institut für Software, Switzerland)
- Jason Yip (ThoughtWorks, Australia)
- Klaus Marquardt (Dräger Medical, Germany)
- Juha Pärssinen (VTT, Technical Research Centre of Finland, Finland)
- Pavel Hruby (CSC, Denmark)
- Lise Hvatum (Schlumberger, USA)
- Ralph Johnson (University of Illinois at Urbana-Champaign, USA)
- Sérgio Soares (University of Pernambuco, Brazil)
- Nobukazu Yoshioka (National Institute of Informatics, Japan)
- António Rito Silva (INESC-ID/Technical University of Lisbon, Portugal)
- Michael Jackson (Independent Consultant, UK)
Patterns for the Secure and Reliable Execution of Processes

Eduardo B. Fernandez
Dept. of Computer Science and Eng.
Florida Atlantic University
Boca Raton, FL, USA
561-297-3466
ed@cse.fau.edu

David L. La Red Martinez
Dpto. de Informática
FaCENA, Univ. Nac. del Nordeste
Corrientes, Argentina
+54-93783-638194
lrmdavid@exa.unne.edu.ar

ABSTRACT
The controlled interaction of processes in a computing environment is fundamental for its security and reliability. Processes can be attacked by other processes or by external clients, errors in one process can propagate to others. We show here three patterns that can help provide a secure and reliable execution environment although they need to be complemented with other patterns. They include Protected Entry Points, which control the correct use of entry points according to their signatures (type and length of parameters); and Protection Rings, which control the calls between processes, enforcing constraints on entry points and signatures according to the level of trust in the processes. Finally, the Multilevel Secure Partitions (MSP) pattern, confines execution of a process to a system partition that has a specific confidentiality or integrity level.

Categories and Subject Descriptors
D.2.11 [Software Architectures]: Patterns (secure & reliable execution of processes).

General Terms

Keywords
Security & Reliability Patterns, Execution of Processes.

1. INTRODUCTION
The In a computer system processes typically collaborate to perform some activity or call each other to request services. Process invocations occur through procedure calls or remote procedure calls; these operations are supported at the kernel level through send/receive operations, which may be direct or indirect (using mailboxes) [1].

The operation name used for invocation, plus the number, type, and length of the parameters in the call is called the procedure signature. The controlled interaction of processes in a computing environment is fundamental for its security and reliability.

Processes can be attacked by other processes or by external clients, errors in one process can propagate to others. Executing processes in a computing system need to be protected from attacks from other processes. Many of those attacks come from the invocation of unprotected (no access control) or wrong entry points or using the wrong type or size of parameters in these calls.

We present here two patterns to prevent these attacks: Protected Entry Points and Protection Rings.

Protected Entry Points control the correct use of entry points according to their signatures. The entry points may apply additional checks, e.g. access control.

Protection Rings control the calls between processes, enforcing constraints on entry points and signatures according to the level of trust in the processes.

Many security attacks propagate through weak parts of a system. After finding an entry point, the malicious software may access a directory or some other unit of the architecture, frequently escalating its power. We need to find ways to stop or obstruct this propagation. We present here a pattern that defines a partitioning of the system with this purpose.

The Multilevel Secure Partitions (MSP) pattern is based on the properties of the multilevel access control model [2]. It confines execution of a process in a system partition that has a specific confidentiality or integrity level. Access from this process to other partitions is performed through specific, protected entry points, restricted according to the rules of a multilevel security model.

Figure 1 shows how these patterns relate to each other.

The Protected Entry Point pattern provides a mechanism for controlling calls between processes.

The Protection Rings pattern and the Multiple Secure Partitions pattern restrict interprocess calls. The MSP pattern uses the multilevel model to control calls.

Section 2 discusses the Protected Entry Points pattern while Section 3 presents the Protection Rings pattern. Section 4 discusses
the MSP pattern and we end with some conclusions and ideas for future work.

Figure 1. Pattern diagram of the patterns in this paper.

2. PROTECTED ENTRY POINTS
This pattern forces a call from a process to another to go through only pre-specified entry points where the correctness of the call is checked and other access restrictions may be applied.

2.1 Example
ChronOS is a company building a new operating system, including a variety of plug-in services such as media players, browsers, and others. In their design, processes can call each other in unrestricted ways. This makes process calls fast, which results in general good performance and everybody is satisfied. However, when they test the system, an error anywhere produces problems because it propagates to other processes, corrupting their execution. Also, many security attacks are shown to be possible. It is clear that when their systems are in use they will acquire a bad fame and they will have problems selling it. We need to have a system which provides resilient service in the presence of errors and resistant to attacks.

2.2 Context
Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. The environment can be centralized or distributed. Some processes may be malicious or contain errors.

2.3 Problem
Process communication has an effect on security because if a process calls another in entry points without appropriate checks the calling process may read or modify data illegally, alter the code of the executing process, or take over its privilege level. If the checks are applied in specific entry points, some languages, e.g., C or C++, let the user manipulate pointers to bypass those entry points. Process communication also has a strong effect on reliability because an error in a process may propagate to others and disrupt their execution.

The solution is affected by the following forces:

- Executing processes need to call each other to perform their functions. For example, in operating systems user processes need to call kernel processes to perform I/O, communications, and other system functions. In all environments, process may collaborate to solve a common problem and this collaboration requires communication. All this means that we cannot use process isolation to solve this problem.

- A call must go to a specified entry point or checks could be bypassed. Some languages let users alter entry point addresses so input checks can be bypassed.

- Typically, a process provides services to other processes but not all services are available to all processes. A call to a service not authorized to a process can be a security threat or allow error propagation.

- In a computing environment we have a variety of processes with different levels of trust. Some are processes which we normally trust, such as kernel processes, others may include operating system utilities, user processes, and processes of uncertain origin. Some of these processes may have errors or be malicious. All calls need to be checked.

- The number, type, and size of the passed parameters in a call can be used to attack a process, e.g., by producing a buffer overflow. Wrong parameters may produce or propagate an error.

2.4 Solution
Systems that use explicit message passing have the possibility of checking each message to see if it complies with system policies. For example, a security feature that can be applied when calling another process is protected entry points. A process calling another process can only enter this process at pre-designed entry points and only if the signature used is correct (name, number of parameters, type and size of parameters). This prevents bypassing entry checks and avoids attacks such a buffer overflows.

2.4.1 Structure
Figure 2 shows a class diagram of the solution. Calling and Called Processes are roles of processes in general. When a Calling Process makes a request for a service to another process, the request is handled by an Entry Point. This entry point has a name and a list of parameters with predefined names, types, and size limits that can be used to check the correction of the call signature. It can optionally add access control checks by using a Reference Monitor pattern or other input data tests.

Figure 2. Class diagram of the Protected Entry Point pattern.

2.4.2 Dynamics
Figure 3 shows a process performing a service call. The call must use a proper signature, i.e., if the name of the service (opName) or the names of the parameters are incorrect, and the type or length of
the parameters is not correct, it is rejected (this is checked by operation checkParmList).

![Figure 3. Sequence diagram for a process making a service call.](image)

2.5 Implementation
As mentioned earlier, kernels support calls as direct calls or through mailboxes. In the first case, the called process must check that the call is correct; in the second case, the mailbox must do the checking.

Entry points must be expressed as references as in Java, and not as pointers as in C or C++ (pointers allow arithmetic operations). In languages that use pointers, it is necessary to restrict their use in procedure calls, e.g. no pointer arithmetic.

2.6 Example Resolved
If parameters of all calls are validated through Protected Entry Points, many security and reliability problems can be avoided. Additional checks, such as access control and data value checks can also be applied.

2.7 Known Uses
- Multics.
- Systems that use ring architectures, e.g. the Intel Series 86 and Pentium.
- Systems that use capabilities, e.g. IBM’s S/6000.
- A specific use can be found in a patent for PC BIOS [3].

2.8 Consequences
This solution has the following advantages:
- If we can check all the calls of a process into another, we can check that the calls are for appropriate services and apply checks for security or reliability purposes.
- Checking the number, type, and length of the parameters passed in a call, can prevent a variety of attacks and stop the propagation of some errors.
- If we know the level of trust of processes we can adjust the number of checks; for example we apply more checks to suspicious processes.

2.9 Related Patterns
- This pattern can be seen as a specific realization of an abstract principle: “Validate input parameters”.
- Protection Rings, see Section 3.
- Multilevel Secure Partitions, see Section 4.
- Capabilities [2].
- Access Control [4] and Distributed Access Control [5]. These checks can be applied in specific entry points to control access to resources.

3. PROTECTION RINGS
Assign processes to a set of hierarchical rings that control how processes call other processes and how they access data. Crossing of rings is done through gates that check the rights of the crossing process. A process calling a process or accessing data in a higher ring must go through a gate.

3.1 Example
The ChronOS designers found that for applications that use programs with a variety of origins, there is a high overhead in applying elaborate checks to all of them. It would be more efficient to apply the checks selectively depending on how much we trust the programs making the calls but we usually don’t know that at execution time. If we could have a way to classify processes according to trust, we could improve the application of checks. Also, we cannot rely on program features to ensure entering the right entry points because applications may come in a variety of languages, some of which may allow skipping entry points.

3.2 Context
Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. Some processes may be malicious or contain errors that may affect process execution. This pattern applies only to centralized environments.

3.3 Problem
Defining a set of protected entry points is not enough if we cannot enforce their use. How can we prevent a process from calling another in an entry point which has no checks? We cannot rely on language features unless we only use a restricted set of languages, not practical in general. If all processes are alike we also need to apply the same checks to all of them, which may be an overkill. The solution is constrained by the following forces:
- We want to be able to enforce the application of Protected Entry Points, at least for some processes. In this way, requests from suspicious processes can be always controlled.
- We would like to separate processes according to their level of trust and check only calls from a low-level to a higher-level process. This can reduce considerable execution-time overhead.
- In each higher-level we want to check signature validity as well as access control or reliability tests. These actions should result in a more secure execution environment.

3.4 Solution
Define a set of hierarchical protection domains, called protection rings (typically 4 to 32) with different levels of trust. Processes are assigned to rings based on their level of trust. Ring crossing is performed through gates that enforce protected entry points: A process calling a higher-level process or accessing data at a higher level can only enter this process or data at pre-designed entry
points with controlled parameters. Additional checks for security or reliability can be applied at the entry points.

3.4.1 Structure
Figure 4 shows a class diagram for this pattern. The Calling Process requests services from a Called Process. To do so, it must enter a Call Gate, that applies Protected Entry Points, that check the correct use of signatures. Call Rules define the requirements for interlevel calls. The Calling Process can access Data according to a set of Data Access Rules.

3.4.2 Dynamics
Figure 5 shows a sequence diagram for a call to a higher privilege ring. If the call fails an exception may be raised.

3.4.3 Variants
Rings don’t need to be strictly hierarchic, partial orders are possible and convenient for some applications. For example, a system including a secure database system could assign a level to this database equal but separated from system utilities; the highest level is for the kernel and the lowest level is for user programs. This was done in a design involving an IBM 370 [6].
In some systems, e.g. the MV8000, rings are associated with memory locations.

Multics used the concept of call bracket. See Section 3.5.

3.5 Implementation
The Call Rules and the Data Access Rules are usually implemented in the Call instruction microcode [7]. Figure 6 shows a typical use of rings. Processes are assigned to rings based on their level of trust; for example, we could assign four rings in decreasing order of privilege and trust to: supervisor, utilities, trusted user programs, untrusted user programs.

The Program Status word of the process indicates its current ring and data descriptors also indicate their assigned rings. The values of the calling and called processes are compared to apply the transfer rules.

The Intel X86 architecture [7] applies two rules:
- Calls are allowed only in a more privileged direction, with possible restriction of a minimum calling level.
- Data at level p can be accessed only by a program executing at a more privileged level (≤ p).

Another possibility to improve security is to allow calls only within a range of rings; in other words jumping many rings is considered suspicious. Multics defined a call bracket, where calls are allowed only within rings in the bracket. More precisely, for a call from procedure i to a procedure with bracket (n1, n2, n3) the following rules apply: if n2 ≤ i < n3 the call is allowed to specific entry points; if i < n3 the call is not allowed, if i < n1 any entry point is valid. This extension only makes sense for systems that have many rings.

3.6 Example Resolved
Now we can pre-assign processes to levels according to their trust. All calls to processes of higher privilege are checked. Processes of low trust get more checks.

3.7 Known Uses
- Multics introduced this concept and used 32 rings as well as call brackets (see Section 3.5) [8].
- The Intel Series X86 and Pentium [7].
- MV8000 [9, 10].
Hitachi HITAC.
Other computers using this idea are the ICL 2900, VAX 11, and MARA, described in [11], which also describes Multics and the Intel series.
[12] shows a use of rings to protect against malicious mobile code.
An IBM S/370 was modified to have non-hierarchical rings [6].
Rings have been used for fault-tolerant applications [13].

3.8 Consequences
This pattern has the following advantages:

- We can separate processes according to their level of trust.
- Level transfers happen only through gates where we can apply Protected Entry Points; that is, we have enforced protected entry points for upward calls.
- We can control procedure calls as well as data access across levels.

Possible disadvantages include:

- Crossing rings take time. Because of this delay some operating systems use fewer rings. For example, Windows uses 2 rings, IBM's OS/2 uses 3 rings [14]. Using fewer rings improves performance at the expense of security.
- Without hardware support the crossing ring overhead is unacceptable, which means that this approach is only practical for operating systems and for centralized environments.

3.9 Related Patterns
A combination (process, domain) corresponds to a row of the Access Matrix [4]:

- Multilevel Secure Partitions (See Section 4). That pattern is an alternative for distributed environments, where processes are assigned levels based on multilevel security models [2].
- Protected Entry Points (See Section 2).

4. THE MULTILEVEL SECURE PARTITIONS PATTERN
Confines execution of a process in a system partition which has been assigned to a specific confidentiality or integrity level. Access from this process to other partitions (processes or data) is restricted according to the rules of a multilevel security model, where processes have sensitivity levels.

4.1 Example
ChronOS is now building a web system. The system's Web, Application, and Database Servers are separated because it is clear that the Web Server has a higher exposure to attacks. If the Web server is compromised, ChronOS will be unable to provide some business services to its clients but at least the hacker is denied immediate access to Application and Database servers where the corporate data is stored. ChronOS wants to limit the damage from any one attack and prevent the attacker erasing their steps from the logs. The normal operation of the application requires processes to request and obtain services between the elements of the system.

4.2 Context
Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. The environment can be centralized or distributed. Some processes may be malicious or contain errors.

In multilevel models data and procedures are classified into sensitivity levels and users have access to them according to their clearances. These models have been formalized in three different ways [2]:

- The Bell-La Padula model, intended to control leakage of information between levels.
- The Biba model, which controls data integrity.
- The Lattice model, which generalizes the partially ordered levels of the previous models using the concept of mathematical lattices.

4.2.1 The Bell-La Padula Confidentiality Model
This model classifies subjects and data into sensitivity levels. Orthogonal to these levels, compartments or categories are defined, which correspond to divisions or groupings within each level. The classification, C, of a data object defines its sensitivity. Similarly, users or subjects in general are given clearance levels. In each level an access matrix may further refine access control.

A security level is defined as a pair (classification level, set of categories). A security level dominates another (denoted as =>) if and only if its level is greater or equal than the other level and its categories include the other categories. Two properties, known as “no read up” and “no write down” properties, define secure flow of information:

Simple security (ss) property. A subject s may read object o only if its classification dominates the object’s classification, i.e., C(s) => C(o). This is the no read-up property.

*-Property. A subject s that can read object o is allowed to write object p only if the classification of p dominates the classification of o, i.e., C(p) => C(o). This is the no write-down property.

This model also includes trusted subjects that are allowed to violate the security model. These are necessary to perform administrative functions (e.g., declassify documents, increase a user’s clearance). A pattern for the Bell-LaPadula model was given in [15].

4.2.2 The Biba Integrity Model
Biba’s model classifies the data into integrity levels (I) and defines two properties dual to the simple security and * properties:

Single integrity property. Subject s can modify object o only if I(s) => I(o).

Integrity *-property. If subject s has read access to object o with integrity level I(o), s can write object p only if I(o) => I(p).

The first property establishes that an untrusted subject cannot write to objects of a higher level of integrity or she would degrade that object.
4.3 Problem
Many security attacks propagate through weak parts of a system. After finding an entry point, the malicious software may access a directory or some other unit of the architecture, frequently escalating its power because it will inherit rights from the compromised units. Protection rings can help with this problem but they can only be applied at the operating system level because of their need for hardware support.

The following forces affect the solution:
- Even if one part of the system is compromised or corrupted, other units will remain unaffected; that is, attacks or errors in a partition at some level should not propagate to other levels.
- The solution should be applicable to all architectural levels of the system, not just the operating system level.
- The solution should be applicable to distributed environments, not just single-processor systems.

4.4 Solution
Divide the architecture in such a way that transfers of control or data access from one division to another follows the Bell LaPadula and/or Biba restrictions. Assign functionality to levels according to their sensitivity.

4.4.1 Structure
Figure 7 shows a class diagram for the solution. A Client Process is assigned by the System to a specific Partition according to their function and degree of trust. A process can call other partitions for services according to a set of Transition Rules, based on a multilevel model.

```
Process * executesIn 1 System

assignment * Partition

* Transition Rule
```

Figure 7. Class diagram for the Multilevel Secure Partitions pattern.

4.4.2 Dynamics
Figure 8 shows a sequence diagram for a process requesting a service from a different partition (p2), than the one where it is executing (p1). Requests for services in other partitions follow the rules of the multilevel model.

4.5 Implementation
The system must support a multilevel access control policy with mandatory restrictions, including data labeling and rule enforcement. Transitions from one partition to another should happen only through Protected Entry Points.

4.6 Example Resolved
Figure 9 shows the solution provided by HP to the ChronOS problem [16]. Now, if the web server is compromised, the attacker cannot deface web pages, cannot attack the web application server, cannot erase the log (they are all in different partitions).

```
<<actor>> :Process
:System

serviceRequest (p1, p2)

evaluateRequest (p1, p2)

performService
```

Figure 8. Sequence diagram for requesting a service within a partition.

```
 outside

Ext. Browser

CGI

Int.

Web Server

Audit Trail

CGI

Inside

Int.

Gateway

Ext. Browser

CGI

Int.

Web Server

Audit Trail

CGI

Int.

Browser

```

Figure 9. HP’s Virtual Vault architecture.

4.7 Known Uses
- HP’s Virtual Vault [17, 18, 16]—This is a secure platform for Internet applications that includes a trusted HP-UX operating system (a Unix variant), an enterprise server, firewalls, and other protection devices. It was part of a secure server system, the HP Praesidium family of products and appears to be the first use of this pattern. It also reduces the root privileges and controls inheritance of rights in forked threads (See Controlled Inheritance of Rights in [19]).
- HP’s restructuring of Unix’ Sendmail program and other system programs [20].
- HP’s UX-11i, the current version of HP’s operating system, also uses this approach [hpux].

4.8 Consequences
This pattern has the following advantages:
A compromised (taken over) or corrupted partition cannot propagate its attack or errors to other partitions.

- The solution does not depend on hardware support and can be applied in any architectural level, from applications to the operating system, to distribution units as in a Service-Oriented Architecture (SOA).
- The solution is particularly suitable for distributed systems because it depends only on local attributes of the procedures and data involved.
- A multilevel model defines the relationship between subject and objects using a lattice model where it can be formally proven that the permitted accesses do not violate security restrictions. This only proves security in an abstract sense and could be affected by implementation details but it is a good security guideline for the complete system.
- Can lead to a more systematic definition of privileges because the levels can help in structuring them. This also makes administration simpler.

Possible disadvantages include:

- It may not be easy or even possible to place the required functions and data in the appropriate levels. This is particularly true at the application level, most commercial environments are not hierarchical.
- It is not simple to change the levels of existing programs or data. A trusted program is needed to override the rules and move subjects or data to other partitions [2].

4.9 Related Patterns

- Multilevel Access Control pattern [4].
- Protected Entry Points, see Section 2.
- Protection Rings, see Section 3.
- Protected Address Space (Sandbox) [19]. The MSP pattern can only control the actions from a process with respect to other partitions, the Protected Address Space pattern can define precisely which resources within a partition can be accessed by the process.

5. CONCLUSIONS

We have presented three patterns that can improve the security and reliability of executing processes. Both the Protection Rings and the MSP patterns use Protected Entry Points to control changes of domain. While MSP applies to the control of executing processes in any level of the system and in distributed systems, Protection Rings require operating system and hardware support and only apply to centralized systems.

6. ACKNOWLEDGEMENTS

We thank our shepherd, Neil Toussaint for his valuable suggestions that significantly improved this paper.

7. REFERENCES