GUIDELINES FOR FINDING DATA CONCURRENCY PROBLEMS IN BUSINESS PROCESS MODELS

by

Kevin Burr

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Applied Science in Technology Innovation Management

Department of Systems and Computer Engineering

Carleton University

Ottawa, Canada, K1S 5B6

May, 2011

© Copyright 2011 Kevin Burr
NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Canada
The undersigned hereby recommend to
the Faculty of Graduate Studies and Research acceptance of the thesis

GUIDELINES FOR FINDING DATA CONCURRENCY
PROBLEMS IN BUSINESS PROCESS MODELS

Submitted by
Kevin Burr

In partial fulfillment of the requirements for the degree of
Master of Applied Science in Technology Innovation Management

Dr. Howard Schwartz, Department Chair

Dr. Michael Weiss, Thesis Supervisor

Carleton University
May, 2011
ABSTRACT

Finding concurrency problems in software can be very hard because the problems may not appear until all of the components have been integrated together and just the right conditions occur. In production, these problems may even seem to be random. In software, these problems have been well researched, however research in Business Process Modeling is still relatively new, and the people who are experts on business processes may not be familiar with all the knowledge contained within the realm of software engineering.

This thesis adapts well known solutions to common data concurrency problems from software engineering, to guidelines business process modelers can use to detect where potential problems may hide and suggest possible solutions. In software engineering terms, the thesis takes the literature on data concurrency patterns and principles and combines it to form a catalog of process model smells (like symptoms) with references back to how the problems should be dealt with.

In order to create these guidelines, a pattern analysis framework is described which amalgamates existing pattern research and re-categorizes it by smells instead of solutions as patterns are typically organized. It also synthesizes new information by revealing new relationships between different research sources.
ACKNOWLEDGMENTS

I would like to thank God,
without Whose help this thesis would not have been possible,

Jenny, Tara and Kacie for their patience,

and Professors Weiss and King for their guidance.
5.4 Opportunities for future research ................................................................. 63
6 REFERENCES ..................................................................................................... 65
A. DATA CONCURRENCY GUIDELINE LIST ....................................................... 72
   A.1 Concurrency: ............................................................................................ 72
   A.2 External: .................................................................................................. 72
   A.3 Internal: ................................................................................................... 73
   A.4 Robustness: .............................................................................................. 74
   A.5 Routing: .................................................................................................. 74
   A.6 State: ...................................................................................................... 74
   A.7 Termination: .............................................................................................. 75
   A.8 Transfer: .................................................................................................. 75
   A.9 Trigger: .................................................................................................... 75
   A.10 Usability: ................................................................................................ 75
   A.11 Usage: ..................................................................................................... 76
   A.12 Visibility: ............................................................................................... 76
B. DATA CONCURRENCY GUIDELINES AND ISSUES (DEPTH OF 3) ......... 77
   B.1 Concurrency: ............................................................................................ 77
      B.1.1 Using a lock when making changes to something shared ............. 77
      B.1.2 Making changes to copied data ..................................................... 80
      B.1.3 Concurrency management ............................................................... 81
      B.1.4 Accessing shared data .................................................................... 84
   B.2 External: .................................................................................................. 86
      B.2.1 Sending data outside of the workflow ........................................... 86
      B.2.2 Receiving data or requests from outside workflow .................... 87
      B.2.3 Retrieving data from outside of the workflow ............................. 89
   B.3 Internal: .................................................................................................. 91
      B.3.1 Data used by multiple instances .................................................... 91
      B.3.2 Separate data and control flow channels ................................... 91
      B.3.3 Merging results from multiple instance tasks ............................. 96
LIST OF FIGURES

Figure 1 Workflow Structure ................................................................. 28
Figure 2 Note Types and Relations .......................................................... 35
Figure 3 Integration Links ................................................................. 36
Figure 4 Node Type Entity Relationship Diagram ....................................... 37
Figure 5 Smell Database Structure .......................................................... 38
Figure 6 Database Entity Relation Diagram ............................................... 39
Figure 7 Graphical Representation of Database ............................................ 46
Figure 8 Depth of 1 - Just Smell Nodes ..................................................... 47
Figure 9 Depth of 3 ............................................................................... 48
Figure 10 Depth of 4 .............................................................................. 49
Figure 11 Depth of 5 .............................................................................. 50
Figure 12 Example Tree - Depth 3 ............................................................ 52
Figure 13 Example Tree - Depth 4 ............................................................ 55
Figure 14 Interactive Guidelines Example .................................................. 59
# LIST OF TABLES

Table 1 Modeling language perspectives ................................................................. 7  
Table 2 Layers of Concern ...................................................................................... 10  
Table 3 Streams about the use of BPM languages ...................................................... 11  
Table 4 Research method ......................................................................................... 33  
Table 5 Database statistics ....................................................................................... 46  
Table 6 Database statistics for depth of 3 .................................................................. 47  
Table 7 Statistics for depth of 4 ................................................................................. 49  
Table 8 Statistics for tree depth of 5 .......................................................................... 50  
Table 9 Statistics for depth of 3 with similar nodes promoted ................................... 51  
Table 10 Statistics for depth of 4 with similar nodes promoted ................................. 51
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>6-Sigma</strong></td>
<td>Business management strategy originally developed by Motorola</td>
</tr>
<tr>
<td><strong>Atomic task</strong></td>
<td>A task which has a self-contained definition and only one instance of the task executes when it is initiated. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>Block task</strong></td>
<td>A complex action which has its implementation described in terms of a sub-workflow. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>BPA</strong></td>
<td>Business Process Architecture (vs. Automation)</td>
</tr>
<tr>
<td><strong>BPEL</strong></td>
<td>Business Process Execution Language</td>
</tr>
<tr>
<td><strong>BPM</strong></td>
<td>Business Process Modeling (vs. Management)</td>
</tr>
<tr>
<td><strong>BPMN</strong></td>
<td>Business Process Modeling Notation</td>
</tr>
<tr>
<td><strong>BPMN-Q</strong></td>
<td>BPMN graphical Query language (Laue et al. 2009)</td>
</tr>
<tr>
<td><strong>BPR</strong></td>
<td>Business Process Re-engineering</td>
</tr>
<tr>
<td><strong>Case</strong></td>
<td>A process instance (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>CBP</strong></td>
<td>Paper used in pattern/smell database: &quot;Concurrent Bug Patterns and How to Test Them&quot;, (Farchi et al. 2003)</td>
</tr>
<tr>
<td><strong>Components</strong></td>
<td>Refers to all of the tasks that comprise a given workflow model. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>Control channel</strong></td>
<td>Supports control flow between tasks. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>Control flow</strong></td>
<td>The order tasks are executed. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>CORBA</strong></td>
<td>Common Object Request Broker (enables distributed software components to communicate with each other)</td>
</tr>
</tbody>
</table>
CPN  Coloured Petri-nets - extension to petri-nets in which every token has a value


Data channel  Provides a means of communicating data elements between two connected workflow tasks. (van der Aalst et al. 2010)

Environment  The world the workflow operates within. It may contain other workflows, processes or data sources that tasks need to communicate with. (van der Aalst et al. 2010)

EPC  Event-driven Process Chain

ITIL  Information Technology Infrastructure Library

Lean  "Lean" production practice, originally from Toyota

Multi-instance block task  is a combination of the two previous constructs and denotes a task that may have multiple distinct execution instances each of which is block structured in nature (i.e. has a corresponding sub-workflow). (van der Aalst et al. 2010)

Multi-instance task  is a task that may have multiple distinct execution instances running concurrently within the same workflow case. (van der Aalst et al. 2010)

PAP  Paper used in pattern/smell database: "Process anti-patterns: How to avoid the common traps of business process modeling" (Koehler et al. 2007)
**Petri-net**  Graphical mathematical modeling language used to describe distributed systems

**PMG**  Seven Process Modeling Guidelines (7PMG), (Mendling et al. 2010)

**PPR**  Portland Pattern Repository

**Process instance**  An executing instance of a workflow model. There may be multiple cases of a particular workflow model running simultaneously. (van der Aalst et al. 2010)

**RPC**  Remote Procedure Call (for inter-process communication)

**SAP**  Commercial enterprise software vendor. Acronym is German for "Systems, Applications and Products in data processing"

**SEQUAL**  "SEmiotic QUALity framework", a systems modeling reference framework for evaluating the quality of models. (Krogstie et al. 2006)

**SME**  Small/Medium Enterprises

**SOA**  Service-Oriented Architecture

**SOAP**  Simple Object Access Protocol for exchanging information for Web services.

**Staffware**  Commercial process modeling tool suite

**Sub-workflow**  When a block task is started, it passes control to the first task(s) in its corresponding sub-workflow. (van der Aalst et al. 2010)

**Task**  Corresponds to a single unit of work. (van der Aalst et al. 2010)
<table>
<thead>
<tr>
<th><strong>Task instance</strong></th>
<th>An invocation of a specific <em>task</em>. (van der Aalst et al. 2010)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TQM</strong></td>
<td>Total Quality Management</td>
</tr>
<tr>
<td><strong>UML</strong></td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td><strong>URN</strong></td>
<td>User Requirements Notation (Weiss et al. 2005)</td>
</tr>
<tr>
<td><strong>VIDE</strong></td>
<td>&quot;VIvisualize all moDel drivEn programming&quot;. (Rech et al. 2007)</td>
</tr>
<tr>
<td><strong>WCP</strong></td>
<td>Paper used in pattern/smell database: &quot;Workflow Control-Flow Patterns: A Revised View&quot; (Russell et al. 2006)</td>
</tr>
<tr>
<td><strong>WDP</strong></td>
<td>Paper used in pattern/smell database: &quot;Workflow Data Patterns&quot; (Russell et al. 2004a)</td>
</tr>
<tr>
<td><strong>WebSphere</strong></td>
<td>IBM WebSphere enterprise software</td>
</tr>
<tr>
<td><strong>Workflow</strong></td>
<td>Short for <em>workflow model</em>. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>Workflow model</strong></td>
<td>A description of a business process in sufficient detail that it is able to be directly executed by a workflow management system. It is composed of <em>tasks</em> which are connected in the form of a directed graph. (van der Aalst et al. 2010)</td>
</tr>
<tr>
<td><strong>WRP</strong></td>
<td>Paper used in pattern/smell database: &quot;Workflow Resource Patterns&quot; (Russell et al. 2004b)</td>
</tr>
<tr>
<td><strong>XPDL</strong></td>
<td>XML Process Definition Language for exchanging business process definitions</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

Business processes have been around as long as there has been commerce, and can involve large numbers of concurrent processors (i.e. people). Likewise, business process modeling can be considered to have begun the first time someone drew a picture to help explain to someone else how their business worked.

Despite this, there is very little literature about verifying Business Process Models (BPM) and where problems may hide, especially when looking for problems with shared data. However, these are well known problems in software engineering, telecommunications, service oriented architectures, etc. Therefore if a large number of people working together can be compared to concurrent processors, then the knowledge from the software engineering world should be applicable to business processes as well.

Business people are not programmers, so this knowledge needs to be intuitive: easy for business people to learn and use. Research in concurrency problems often uses finite state machines and particularly coloured petri-nets. These are not described in a way most business people would understand and are therefore not very usable.

In software, patterns are proven solutions to common problems. Smells are indicators of code that may hide problems and may need to be re-factored (re-designed) using the correct patterns.
1.1 Objectives

The objective of this research is to move modeling knowledge from software engineering to the business domain and thus make it available to business process modelers.

1.2 Deliverables

The deliverables of the research are:

1. An initial set of guidelines of where to find concurrency-related business process modeling data issues and pointers to possible solutions in literature.

2. A catalog of old and new relationships between smells, issues and patterns, with links back to their original source.

3. Prototype tools for capturing patterns, issues and smells from literature and analyzing their inter-connections.

Note that the guidelines in this thesis are examples of what can be created using this framework. This is due to the amount of software engineering literature that would need to be analyzed and the lack of related work to build upon. The validation of the guidelines is also left as future work. Additionally, while most guidelines suggest solutions, it is up to the modeler to decide which best suits their circumstances, based on the referenced descriptions.

1.3 Relevance

This research is relevant to at least three groups.
1 Top Management Teams need processes to be precisely documented. In a global economy where services can be provided anywhere in the world, the competitive market requires higher quality at a lower cost and cycle time (Przyblek, 2007). Business process modeling is becoming more common to help merge activities around the world. Modeling guidelines will then make it easier to accurately identify and optimize processes and thereby reduce product cycle times. It is far too late to discover there are timing and synchronization problems after a process has been deployed around the world.

2 Business Managers and Analysts are the people who typically design and manage these processes. The ability to find problems and disconnects between different business groups by validating processes and interactions before implementing them across a corporation will improve their chances of success and ability to influence change. (Kazanis & Ginige, 2002) Even just a document capturing possible problems and how to avoid them will be extremely valuable to them. BPM can also be used for requirements analysis, and to generate testcases (Amyot et al. 2005).

3 Finally, Researchers will find this study useful. Research in Business Process Modeling in general is still relatively new and only covers a small part of the possible issues (Larsen et al. 2005). This thesis also provides a way to look at old knowledge in software engineering in another domain in a novel way which may provide inspiration for further research.
1.4 Contributions

1 In Software Engineering terms, “Patterns” can be described as well-known solutions to common problems. The process of codifying knowledge used in the thesis when applied to patterns makes the following contributions:

   a. Patterns are usually organized by the types of solution they provide. Reorganizing this information into “Smells” (i.e. guidelines for detecting situations that may hide errors) provides a novel way to recognize problems in business process models.

   b. Breaking patterns from different research sources apart and then recombining them as smells synthesizes new information and reveals relationships between the different sources that were not documented before. This process could be applied to other knowledge domains as well.

2 The proof of concept guidelines help BPM practitioners to improve the quality of their models, through the description of how to avoid/detect data concurrency problems, which were well known in Software Engineering but unfamiliar to business process modelers.

1.5 Organization

This thesis is organized into five chapters, and three appendices. Chapter 1 is the introduction. Chapter 2 covers the review of the relevant literature. Chapter 3 presents the method used to conduct the research. Chapter 4 contains the research results. Chapter 5 provides the conclusions, describes the limitations of the research, and identifies opportunities for future research. Appendix A presents a summary list of data
concurrency guidelines derived from the sample of related literature, and finally appendix B contains guidelines created by drilling three levels into the patterns database.
2 LITERATURE REVIEW

This entire thesis is a literature review about data concurrency issues in business process modeling. Chapter 3 will discuss how the literature was chosen and analyzed, then chapter 4 and the appendices will discuss the results in detail. Therefore, this section will review how this topic fits into the current research.

This chapter is divided into six sections: A brief review of the literature on business process modeling languages in general, and then on how BPM languages are used. The next two sections review first how process models are verified, and then how knowledge from other research areas can be used. The final section discusses lessons learned from the literature.

2.1 Business Process Modeling

2.1.1 History

Almost every paper on Business Process Modeling, Management, Improvement, or Re-engineering has a section on their inter-connected roots. This will not be repeated here, other than to mention BPM’s roots are usually traced back to the early 20th century as a way to improve how organizations work. The 1970s onwards saw the birth of office automation and in the 1990s the growing popularity of Business Process Re-engineering (BPR, as well as TQM, 6-sigma, etc.) all with associated charts and modeling methodologies. (Mendling et al. 2010)

In this century, interest in BPM continues to grow, as does the creation of a multitude of tools and languages. Yet the more formal modeling languages still have not completely
taken off (Russell et al. 2004a), and most business modeling is still done in tools like PowerPoint, Word, Visio and the like.

2.1.2 Languages

Modeling how a business works (or should work) has often involved static diagrams, like fishbone diagrams, pareto diagrams, quality function charts, etc (Kettinger et al. 1997). One of the main reasons interest in BPM languages is growing is that it allows both a consistent way to capture processes and improve the chances of finding problems before new processes are deployed. (Mendling et al. 2010) (Harmon et al. 2010) BPM languages are also becoming able to automate the processes, either directly or through translation.

<table>
<thead>
<tr>
<th>Role</th>
<th>Perspective</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>What</td>
<td>Functional</td>
<td>What process elements are performed</td>
</tr>
<tr>
<td>When</td>
<td>Behavioral</td>
<td>When process elements are allocated (e.g. sequencing), and how related actions are performed.</td>
</tr>
<tr>
<td>Where</td>
<td>Organizational</td>
<td>Where and by whom in the organization process elements are performed.</td>
</tr>
<tr>
<td>Who</td>
<td>Informational</td>
<td>Entities produced by a process, such as data, documents, etc.</td>
</tr>
<tr>
<td>How</td>
<td>Validation and Verification</td>
<td>How the model was tested to do what it is supposed to do the way it is supposed to do it.</td>
</tr>
<tr>
<td></td>
<td>Modeling procedure</td>
<td>How the model was built</td>
</tr>
</tbody>
</table>

Table 1 Modeling language perspectives (from Lin et al. (2002))

Most modeling languages are graphical and can be considered the decedents of flowcharts from the 1970s and other earlier state diagrams. These languages have different strengths and weaknesses when capturing processes, which Curtis et al. (1992),

Academic literature contains the largest number of process representations, either in use or just proposed. Petri nets are most commonly used for the underlying model or at least a comparable token based language (Trčka et al. 2009). “A Petri net is a graphical tool for the description and analysis of concurrent processes which arise in systems with many components (distributed systems).”\(^1\) Petri nets are mathematically rigorous, and there is a great deal of research available about using it for process analysis.

In industry literature, BPMN is the most commonly used notation. (Laue et al. 2009) (Harmon et al. 2010) Officially: “The Business Process Modeling Notation (BPMN) is a graphical notation that depicts the steps in a business process. BPMN depicts the end to end flow of a business process. The notation has been specifically designed to coordinate the sequence of processes and the messages that flow between different process participants in a related set of activities.”\(^2\) BPMN has a token based foundation like Petri nets, but is targeted towards business users and is focused on ease of use. This has lead

---

\(^1\) The graphics, together with the rules for their coarsening and refinement, were invented in August 1939 by the German Carl Adam Petri - at the age of 13 - for the purpose of describing chemical processes


http://www.scholarpedia.org/article/Petri_net

some researchers to be concerned that its lack of mathematical rigor will make some
types of errors much harder to find (Börger et al. 2008). As an example, BPMN does not
define to what degree it supports multiple tokens, so in the case of synchronizing tokens
(N-of-M OR-join) from an earlier multiple choice split (N-of-M OR-split) it does not
specify how many levels of splits and joins are supported before there are too many
tokens to keep track of. Once tokens are lost, the process will hang waiting for events
that can no longer be recognized.

2.1.3 Layers of Concern

Business Process Modeling means different things to different people. As part of
preparing for this thesis, people who considered themselves as doing business process
modeling were spoken to informally. It turned out, the people found worked on a wide
variety of modeling endeavors, from high-level business reengineering (e.g. using MS
Word as their modeling software) to people working on UML business rules for Service
Oriented Architectures, to BPMN-based simulations and finally to paper-based flow-
charts for training development.

Just as Lin et al. talked about different BPM languages having different strengths and
weaknesses (Lin et al. 2002), Harmon talked about different parts of an organization
having different purposes (or concerns) when building models (Harmon 2007). He
portrays this as a pyramid to represent the increasing number of people and roles involved
from higher to lower levels. This has been greatly simplified in Table 2.
At the top, at the Organizational Level, modelers want to capture organizational goals and measure/improve the organizations performance and compliance. At the Business Level, management wants to create/capture the processes specific to their projects. Finally at the Implementation level, Human Resource and IT Development must build the infrastructure needed (e.g. training, documentation management, etc.) to make these processes work. The models needed at each level are different in their level of abstraction and functionality.

<table>
<thead>
<tr>
<th>Level</th>
<th>Driver</th>
<th>Focus</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise</td>
<td>Operations</td>
<td>Process Improvement</td>
<td>TQM, 6 Sigma, Lean</td>
</tr>
<tr>
<td></td>
<td>Research &amp; Quality Control</td>
<td></td>
<td>…</td>
</tr>
<tr>
<td></td>
<td>Redesign</td>
<td>Process Redesign</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>Information Technology</td>
<td>Process Automation</td>
<td>BPMN, SAP, Staffware</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td></td>
<td>BPEL, XPDL, SOA, Web Services</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UML, URN, SOAP, RPC, CORBA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Networks, Computers</td>
</tr>
</tbody>
</table>

Table 2 Layers of Concern (adapted from Harmon (2007))

The focus for this thesis was chosen to be the level where workflow modeling takes place. This is low enough to take advantage of the concurrency lessons learned from Software Engineering in developing SOA, Realtime Systems, etc., but high enough that
the people doing the modeling are likely coming from a business background and
unaware of this knowledge and its applicability to their tasks.

2.2 Using a Business Process Modeling Language

<table>
<thead>
<tr>
<th>Stream</th>
<th>Key highlights of the stream</th>
<th>Key references</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capturing Business Processes</td>
<td>• representing business processes in models</td>
<td>• Eriksson &amp; Penker (2000)</td>
</tr>
<tr>
<td></td>
<td>• options as UML, BPMN, other methods</td>
<td>• List and Korherr (2005)</td>
</tr>
<tr>
<td>Business Automation</td>
<td>• using model in the development of IT / software solutions to improve business operations</td>
<td>• Chung &amp; Pak (2006)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Chowdhary et al. (2006)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• de Castro (2006)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Nasiri (2004)</td>
</tr>
<tr>
<td>Verification and Validation</td>
<td>• verification / testing of models</td>
<td>• Chowdhary et al. (2006)</td>
</tr>
<tr>
<td></td>
<td>• showing the validity / accuracy of a model - its ability to model the real world “adequately enough”</td>
<td>• Fishwick (2007)</td>
</tr>
<tr>
<td></td>
<td>• simulate the business process, through iterations of the models, through time</td>
<td>• Hseuh et al. (2008)</td>
</tr>
<tr>
<td>Deployment and Execution</td>
<td>• methods and importance of implementing what has been developed or learned</td>
<td>• Chung &amp; Pak (2006)</td>
</tr>
<tr>
<td></td>
<td>• includes distribution, training, measurement, and maintenance aspects as well</td>
<td>• Chowdhary et al. (2006)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Machado et al. (2005)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Nasiri (2004)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Hseuh et al. (2008)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Kazanis &amp; Ginige (2002)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Przyblek (2007)</td>
</tr>
</tbody>
</table>

Table 3 Streams about the use of BPM languages

There is a great deal more literature on how to capture business processes and how they
are applied in different industries than there are papers on verification and validation,
showing that this is an area often overlooked. This section is divided into four streams: Capturing Business Processes, Business Automation, Verification and Validation, and finally Execution and Deployment.

2.2.1 Capturing Business Processes

In order to discuss how to model a business process, you must first understand what a business process is. For example, one of the key works on UML for BPM is by Eriksson and Penker (2000). They built upon the business process foundations of Davenport, Hammer and Champy in the early 1990s, and derived the following definition of a business process. A business process is something that (Eriksson & Penker, 2000):

1. Has a goal.
2. Has specific input.
3. Has specific output.
4. Uses resources.
5. Has a number of activities that are performed in some order, depending on conditions and events that occur during the execution of the process. The activities within the process can be seen as sub-processes.
6. Affects more than one organizational unit. It is horizontal rather than vertical in regard to the traditional organization of the business.
7. Creates value to some kind of customer. The customer can be either internal or external to the business.

As with many researchers, List and Korherr (2005) built upon the work done by Eriksson and Penker, and they describe how they have customized the Eriksson-Penker UML
Extensions in their environment. List and Korherr offer a useful summation of what the requirements of a business process are, in order to be able to model it. This includes the ability to represent activities, roles, customers, process owners, process types, deliverables and measures.

2.2.2 Business Automation

Nasiri (2004) points out that “Rarely there is a business system capable of being fully automated. In other words some parts of a business system always remain manual because of their nature. The intelligence to recognize which parts should be automated and which should be left intact is an artistic activity far beyond current methods.”

BPMN is able to generate BPEL to assist with process automation. Other approaches to automating model based business processes are also mentioned in the literature: De Castro (2006) discusses in her paper the stages in which a UML process model can be translated into a web services composition language for generating web based applications. Chung et al. (2006) used it to create a knowledge-based support system (i.e. expert system). Chowdary et al. (2006), describes automatically generating code from UML diagrams to capture business performance metrics.

2.2.3 Verification and Validation

It is important to ensure that the business process model is correct. Research in this stream of literature discusses the verification and validation requirements of a given model or system. Verification and validation (V&V) are key steps to ensuring that the
modeling effort is successful and often ignored or given lip service. As mentioned above, the majority of modeling articles spoke little of the validation and verification.

This subsection talks about the purpose of the different phases of checking a process' quality. Section 2.3: “Quality of Business Process Models” discusses in more detail the different ways this can be accomplished.

**Verification**

Verification is the process of testing truth against requirements, or more specifically, a model is verified that it is built correctly against how it was intended / specified (Fishwick, 2007), i.e. free of bugs / errors / omissions. This is often associated with testing of an executable application but is also necessary for manual processes. Verification is something that must be considered throughout the development of a product, process, model or system. Methods of verification continue to be refined, and advanced modeling and simulation environments often offer automated or assisted means of verification. Hseuh et al. (2008) describe their use of the Object Constraint Language to automatically verify their process models.

**Validation**

Validation is the process of testing system behaviour against empirical observations (Fishwick, 2007), i.e. does the system adequately represent the real world? Validation of models or systems is often neglected, and a system with attractive graphics and appeal, can easily distract from validating that a given model is actually accurate. Chowdhary et
al. (2006) particularly noted that they experienced difficulty with model validation when dealing with complex systems. Hseuh et al. (2008) found that it was effective to have the end-users of a process use a simulation environment to validate if the models were realistic. While it is generally noted that validation is an important step, few of the initially reviewed papers made mention on how it was done, or if it was done at all.

2.2.4 Deployment and Execution

Once business processes have been correctly captured, optimized, validated, and verified, they need to be reintroduced into the business environment either through paper or automated tools. These tools need to be deployed, the people involved need to be trained, and conformance needs to be tracked. “Distribution, which is going to be a common feature of many enterprises, is very seldom taken into account while modeling, which in turn hinders the success of modeling approach”. (Nassiri 2004)

Distribution

Kazanis & Ginige (2002) discuss that different sized businesses will use and deploy BPMs differently. “Big Business” will use models to develop large scale projects and have the resources to go through a massive development phase with large amounts of process automation, followed by a massive training phase and then be able to provide a massive amount of support to keep everything working. In contrast, Small Medium Enterprises (SMEs) have time and budget constraints. They may only make improvements to existing processes which are most likely paper based. Training and support would be provided through word of mouth as opposed to training material.
Training

Business process modeling languages are not always easy to understand, or instead lack too much information to be automated. BPMs based on UML allow BPM experts to talk the same “language” as software experts making it easier to capture design processes, and for the software professionals to automate those processes in turn. It also makes it easier for software engineers to learn new processes when they are introduced. (Przybylek 2007) Ease of understanding is even more important when businesses span the globe and everyone does not speak the same language. Process mistakes can cause costly errors in both time and money.

Both Hsueh et al. (2008) and Machado et al. (2005) mentioned using model driven simulations (or animations) to help people unfamiliar with the modeling language validate if processes had been captured correctly. Hsueh et al. (2008) takes this further and uses the simulation as a way to train new participants, and found that people actually liked learning using a simulator. This proved successful enough in their experiments to be given permission to go to a trial deployment within the financial institution where they were working.

Measurement

As the adage goes, “If you can’t measure it, you can’t manage it”. For process improvement to be effective it is necessary to be able to determine if a new process is being followed and is achieving the desired results. While Chowdhary et al. (2006) discuss automatically generating Business Performance metrics from architectural UML
diagrams, this technique should also be applicable to measuring automated Business Processes captured in UML. It is not obvious from Chowdhary's article if IBM has actually done this. The article does mention that IBM's tools automate creating the code to capture these metrics without needing programmers.

2.3 Quality of Business Process Models

The section above discussed the importance of Business Process Modeling, how it applies throughout and organization’s structure and how it may be used. Quality then becomes of growing importance in order to reduce the chances a mistake may not be found until a process is implemented (potentially around the world), as well as reducing the accompanying economic cost.

2.3.1 Testing

When a business process is fully automatable, then it can be tested similar to a software program and one can possibly use the model to generate testcases to use on the final implementation (Amyot et al. 2005). However, as Nasiri et al. (2004) point out above, business models are rarely fully automatable. Fortunately, modeling tools often provide the ability to test the model through simulation (Hsueh et al. 2008).

Since it is unlikely to be practical (or even probable) to test every possible execution pathway through the model, then testers have to use experience to know where to look for problems, or depend on tools to look for them (if such tools exist).
2.3.2 Analysis Techniques

Over the last 20 years researchers have been working on tools to analyze the quality of models based on structural metrics. Van der Aalst et al. (2008) provide an overview of these techniques, but also point out that most of these are based on AND/XOR graphs or the equivalent control flow in more sophisticated modeling languages (e.g. Petri-nets, BPMN, EPC, etc.).

In particular, van der Aalst et al. mention three techniques: Coverability graphs, which indicate whether the entire graph is reachable, as well as potential deadlocks and infinite loops. Invariants, where the absence of certain place and transition invariants indicate possible problems. Finally, there are Reduction rules, which simplify the net without removing important structure so the other approaches can execute in a reasonable amount of time.

Mendling et al. (2008) demonstrate using reduction rules to analyze 600 publically available SAP reference models, showing 34 of them contained errors. Laue and Awad (2009) use a graphical query language (BPMN-Q) on 109 models from a public repository to find BPMN model fragments that match structural error patterns (which can be considered a reduction like approach) and found 14 erroneous models.

Analysis problems

There are also concerns with the analysis approach. Koehler and Vanhatalo (2007) point out that the success of most business processes are measured by economic indicators such as cost and profit, and that structural analysis tools can only provide “insights” that
problems may be present. Even worse, these tools' results can be misleading by reporting issues that are not important while completely missing others that are.

Koehler’s and Vanhatalo’s paper (2007) mentions they were not able to find very many papers which address the measurement of quality in process models. A number of papers refer to the Guidelines of Modeling (GoM) framework (Becker et al. 2000), which defines six “guidelines”, correctness, relevance, economic efficiency, clarity, comparability, and systematic design. The SEQUAL framework (Krogstie et al. 2006) lists: physical, empirical, syntactic, semantic, pragmatic, social, and organizational quality types. Finally, in the ISO/IEC 9126 Software Product Quality Model, the quality of process models is based on functionality, reliability, usability, and maintainability. (Guceglioglu & Demirors, 2005) While problems related to attributes like correctness may be derived by analyzing the model, other attributes, like clarity and economic efficiency are more subjective.

Mendling et al. (2010) point out that the level of modeling competence that many casual modelers have is notoriously low. Frameworks like SEQUAL and Guidelines of Modeling tend to be too abstract for non-experts to use, and the various collections of hints lack the sound research to back them up.

Finally, analysis tools are not available for all Business Process Modeling languages. Several studies show small to medium size companies often use “paper based” process modeling due to not having the time and resources for people to become proficient at using formal languages and tools (see Kazanis et al. (2002), Davies et al. (2006)).
2.4 Patterns as Guidelines

As mentioned above, modeling tools are mostly used in large companies (Kazanis et al. 2002) (Davies et al. 2006), and even then business modelers may not have much software engineering (or mathematical) experience with modeling. Guidelines are needed to help them learn the best techniques to do their work.

Two of the better known frameworks: Guidelines of Modeling (GoM), which is inspired by general accounting rules (Becker et al. 00), and SEQUAL, which stands for the “semiotic quality” framework (Krogstie et al. 2006), have proven valuable, but both require a minimum level of expertise to understand and use.

Mendling et al. (2009), in their paper “Seven Process Modeling Guidelines (7PMG)”, provided an overview of the types of guidelines available in business process modeling research and noted the need for more pragmatic and easy to understand guides. They analyzed the available empirical studies and provided a list of seven guidelines backed by research.

These problems have existed in software engineering for decades, and while coding guidelines are still important for consistency, the “best ways” to solve and/or find problems are described in the next three sections: Patterns, Anti-patterns (etc.) and Smells.

2.4.1 Patterns

Patterns are used in software engineering to describe reusable solutions for common problems. This definition appears in multiple places (e.g. Laue et al. 2009). Patterns are
usually captured in a semi-formal format that describes the “solution to a problem in a context” (Lea 2000). Collections of related patterns are sometimes referred to as “pattern languages”. For the purpose of this thesis, patterns are essentially guidelines for ways to solve a problem under specific conditions. They are typically not tied to any programming or modeling language.

Patterns originally came from the world of architecture (e.g. buildings), as reusable descriptions in books by Alexander et al. (1977, 1979). At OOPSLA’87, Kent Beck and Ward Cunningham presented their work applying this idea to programming (Cunningham & Beck, 1987). Software patterns gained in popularity when Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (aka “The Gang of Four”) published the book “Design Patterns: Elements of Reusable Object-Oriented Software” in 1994. Since then, there has been a plethora of papers, books and websites, describing patterns in all areas of design and architecture (not just software, or buildings). More information and references can be found in the Patterns Discussion FAQ (Lea 2000).

For Service Oriented Architecture (SOA) and Web Services and lower layers of concern (section 2.1.3), there is an abundant source of pattern languages and repositories³, many of which contain patterns related to implementing business process logic. Within the workflow modeling community the early major repository of patterns was the Workflow Patterns initiative (van der Aalst et al. 2010) created by WMP van der Aalst and AHM ter Hofstede, primarily as a way to compare workflow modeling tools and languages. The 20

³ and catalogues of repositories. See http://hillside.net/patterns/patterns-catalog, and http://www.dmoz.org/Computers/Programming/Methodologies/Patterns_and_Anti-Patterns
patterns documented in their paper (van der Aast et al. 2003) have been reused in many other papers to introduce or analyze modeling languages. These patterns have expanded to far over a hundred, 126 of which were analyzed as part of this thesis. (Russell et al. 2004a, 2004b, 2006)

2.4.2 Anti-patterns, Error Patterns, etc.

While patterns captured common solutions to common problems, they also began to be used as a way to describe common bad solutions as well. Andrew Koenig coined the term “anti-patterns”, inspired by the Gang of Four’s book (Gamma et al. 1994) and described it as: “An anti-pattern is just like pattern, except that instead of solution it gives something that looks superficially like a solution, but isn’t one” (Koenig 1995). The term was made popular by the book “AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis” (Brown et al. 1998). This book dealt a great deal with aspects of bad management, which actually places it into the management and organizational layers of concern (from section 2.1.3), which is above the level this thesis is focusing on.

Lists of anti-patterns are only one source of common problems. The VIDE Consortium did an exhaustive survey of the literature up to 2007 on quality defects in model driven software (Rech et al. 2007) of approximately 950 papers. From this they created a list of related topics: “Anomalies, Anti-guidelines, Anti-patterns, Bug Patterns, Bug Taxonomies, Critic Rules, Defects, Defect Patterns, Design Bugs & Errors, Error Patterns, Fault Patterns, Flaws, Heuristics, Illnesses, Metric Thresholds, Negative Patterns, Pitfalls, Design Principles, Puzzles / Puzzlers, Design Rules, Code Sins, Styles, Conventions, and Rules” which they document in their report. (Rech et al. 2007)
For workflow models, most papers list the first categorization of error patterns as having been done for deadlock patterns by Onada et al. (1999) at the University of Osaka. Liu and Kumar (2005) discuss how unstructured workflows can be mapped into structured ones, and name the combinations of splits and joins that can lead to control flow errors. Koehler and Vanhatalo (2007) report on lessons they learned reviewing hundreds of process models created in IBM Websphere, as well as few others. All of these are primarily based on control flow.

2.4.3 Smells

...smells represent problematic parts of the software system that seem wrong, complicated, or cumbersome to an experienced developer. In general, smells are problems that are associated with one or more specific refactorings (i.e., concrete treatments) that might be applied to remove the smells. (Rech et al. 2004)

Software smells (unlike Anti-patterns, Error patterns, and others above) do not claim that there is a problem in the code, only that based on experience a senior programmer would be worried that the code may hide problems, be hard to verify, or be hard to make changes to in the future.

The term was coined by Keith Beck in the Portland Pattern Repository [PPR] and made popular in the article “Bad Smells in Code” (Fowler et al. 1999). As the quote above mentions, in general smells are associated with refactoring code (i.e. ways to rewrite/improve/simplify it) in order to improve its testability and maintainability. Code can also become smellier over time, i.e. as more changes are made to code it may become
less maintainable and therefore its smell may give the maintainer hints at how to best refractor it.

There are now a number of lists of smells. The VIDE D4.1 report references several lists (Rech et al. 2004), as does the Portland Pattern Repository [PPR]. Mäntylä (2003) improved the categorization of Fowler's original list to make it easier to use.

In modeling, there are a number of tools which use templates (along the lines of BPMN-Q above) (Laue et al. (2009), Gruhnet al. (2009), Braem et al. (2006), etc), or structural metrics (Mendling et al. (2007), Held et al. (2008), etc) to find smells automatically. While structural metrics (like the number of branch points) may or may not be obvious problems to an experience modeler, they do indicate the potential for problems and are therefore smells. All of these primarily deal with control flow, and if they are not included in a modeler's design tool (if they even have one) they are then of limited value.

The only paper found close to this thesis’ topic was “Refactoring Process Models in Large Process Repositories” by Weber and Reichert (2008). They created a list of eleven refactoring techniques for process models based on software engineering research. Unlike this thesis, they focus on how to fix problems, while this thesis focuses on how to find them. They also limit themselves to control flow, while our focus is data flow, specifically with regards to concurrency.

2.5 Problem Focus

There are an immense number of patterns and associated problem types that guidelines could be built for (e.g. Rech et al. (2007)), which would take far more time and space
than available. As mentioned above, most research focuses on control flow. There is also a lot of information on resource patterns hidden within project management literature.

Therefore this thesis focuses on the convergence of two types of problems that have not been widely studied in BPM: Data and Concurrency, which are discussed in the next two sub-sections. The third sub-section provides common terminology from the literature to describe BPM workflow structures.

2.5.1 Data

Data flow problems occur when values are: over-written un-intentionally, not visible where needed, cause the wrong control flow path to be taken, etc. (See Section 4: RESULTS for a longer list)

Most of the literature about workflow patterns (or anti-patterns, etc.) dwells primarily on control flow. Russell et al. (Russell et al. 2004a) in the largest collection of workflow data patterns (and one of the only) mentions when commenting on existing tool offerings: "One of the most immediate [observations] is that the level of direct support for data patterns in standard workflow design tools is minimal."

The other data flow papers focused on tools or algorithms to find problems. These papers also mentioned the lack of data flow research in business processes.

- Meda et al. (2010) list verification algorithms for data flow, but comment: "Existing workflow systems that model business processes, [...] are generally based on control
flow modeling, but usually also allow input and output data elements to be specified. However, data flow errors are not typically verified.”

- Trčka et al. (2009) give a list of the current research on dataflow analysis, but point out: “Despite the abundance of analysis techniques to discover control-flow errors in workflow designs, there is hardly any support for data-flow verification. Most techniques simply abstract from data, while data dependencies can be the source of all kinds of errors.”

- Sundari et al. (2007) point out: “While a number of analytical and algorithmic methods exist for verifying control-flow correctness, relatively few schemes are available for verifying data-flow correctness.”

2.5.2 Concurrency

Bugs in concurrent programs can be difficult to find and reproduce (Barland et al. 2005), and as stated before, business processes may involve hundreds of individual human “processors” working in parallel. Solutions of how to deal with concurrency problems (e.g. deadlock, race conditions, etc.), and the issues that arise when applying those solutions (e.g. live-lock, starvation, etc.) are common in software engineering, telecommunications, etc., but not well documented for business processes.

Petri nets, which are often used in academia to study process models (Trčka et al. 2009), were invented to deal with parallelism. As mentioned above, most of the research into process modeling has focused on control flow issues like deadlocks (Meda et al. 2010). Russell et al’s (2004a) collection of BPM data patterns list problems with contention for shared data and how to avoid it by serialization, but does not go further and discuss the
problems with serialization. While this knowledge is reasonably well know to software engineers at lower layers of concern, it is not reasonable to expect business-focused modelers (using commercial tools or not) to be aware of this.

Trčka et al. and Russell et al. both have commented on this:

“In the field of software verification, static analysis of software and model checking have been successfully used to discover program bugs that are caused by, e.g., non-initialized or dead variables. In this, totally different, application domain, concurrency issues are rarely treated and systematic classification of errors is missing.” (Trčka et al. 2009)

“Another observation is that many systems/standards do not offer support for multiple instance tasks. A number of the offerings examined implement a shared repository for data elements (at block, scope or case level) where tasks access a common set of data elements rather than passing them between tasks.” (Russell et al. 2004a)

### 2.5.3 Business process model workflow structure

As much as possible, this thesis tries to avoid technical jargon with which the people doing business process modeling may not be familiar. However, in the list of guidelines, there will sometimes be the need to be precise and for that the definitions adapted from van der Aalst’s and ter Hofstede’s Workflow Patterns Home Page (van der Aalst et al. 2010) (Russell et al. 2004a) will be used.

A **workflow** or **workflow model** is a description of a business process in sufficient detail that it is able to be directly executed by a workflow management system.
A workflow model is composed of a number of tasks which are connected in the form of a directed graph.

The environment is the world the workflow operates within. It may contain other workflows, processes or data sources (e.g. people) that tasks need to communicate with.

A case or process instance is an executing instance of a workflow model. There may be multiple cases of a particular workflow model running simultaneously, however each of these is assumed to have an independent existence and they typically execute without reference to each other.

A task instance is an invocation of a specific task. A task instance may initiate one or several task instances when it completes. This is illustrated by an arrow from the completing task to the task being initiated e.g. in Figure 1, task instance B is initiated when task instance A completes. This may also occur conditionally.
A task corresponds to a single unit of work. Four distinct types of task are denoted: atomic, block, multi-instance and multi-instance block. We use the generic term components of a workflow to refer to all of the tasks that comprise a given workflow model.

An atomic task is one which has a simple, self-contained definition (i.e. one that is not described in terms of other workflow tasks) and only one instance of the task executes when it is initiated.

A block task is a complex action which has its implementation described in terms of a sub-workflow. When a block task is started, it passes control to the first task(s) in its corresponding sub-workflow. This sub-workflow executes to completion and at its conclusion, it passes control back to the block task, e.g. block task C is defined in terms of the sub-workflow comprising tasks, X, Y and Z.

A multiple-instance task is a task that may have multiple distinct execution instances running concurrently within the same workflow case. Each of these instances executes independently. Only when a nominated number of these instances have completed is the task following the multiple instance task initiated.

A multi-instance block task is a combination of the two previous constructs and denotes a task that may have multiple distinct execution instances each of which is block structured in nature (i.e. has a corresponding sub-workflow).

The control channel supports control flow between tasks, which is illustrated in Figure 1 by the solid arrows between the tasks.

The data channel provides a means of communicating data elements between two connected workflow tasks. This may be combined with the control channel or may be distinct. In Figure 1, a distinct data channel is represented as the dashed line between task instances C and E.

(van der Aalst et al. 2010)
2.6 Lessons learned

- Business processes suffer from the same problems as distributed computer systems, even when the work is being done by humans

- There is an immense amount of research about data concurrency problems in computer science, but very little corresponding research for business processes

- There is very little research about “smells” as well but there is a lot of related research that can still be used

- If business processes have the same problems as distributed computer systems, then the same strategies to find and prevent these problems should be applicable to business processes as well.
3 RESEARCH METHOD

This thesis uses a constructive approach by building a framework with which to analyze software engineering research in data concurrency problems and convert it into guidelines useable for business modelers to recognize problems.

This chapter describes the methods used to produce the deliverables of this research. It is organized in four sections. Section 3.1 describes the unit of analysis. Section 3.2 describes the theoretical framework. Section 3.3 identifies the research method. Finally, section 3.4 describes how the data was analyzed.

3.1 Unit of Analysis

The unit of analysis was data concurrency pattern descriptions from which issues and solutions were derived. Initially known BPM patterns were used, but the search was expanded into other areas of research, such as Software Engineering. Other sources of problem descriptions, such as those discussed in section 3.4 “Data Acquisition” below, were also investigated.

3.2 Theoretical Framework

Bouhours et al. (2009) described a Design Pattern as “the optimal reusable micro-architecture for a type of problem” and Fowler et al, (1999) states “Smells are any symptom that possibly indicates a deeper design problem”. Therefore this thesis’ “guidelines for finding problems” topic could be referred to as “Smells” in Software Engineering. Bouhours et al. (2009) also show in their paper that an inappropriate or
poorly implemented pattern can be considered a Smell as well. This was used to create a library of smells from pattern and related research. (See section 3.4 below.)

Linda Rising created a Patterns Almanac in 2000 containing summaries of documented patterns up to that date, with links to where the full description can be found. The format is sometimes referred to as a “patlet” (Cunningham 2005), and specifically the format as described in the PatternForge wiki is used:

“A good patlet is short, only about 2-3 sentences. It distills the pattern to its essence. Here, we extend the basic patlet format to include links to related patterns. Each link should document the reason for applying the other pattern.” (Weiss 2007a).

A theoretical framework provides a lens through which to look at the research problem. The concept of "smell" provides a lens through which to look at patterns, and Rising’s framework provides an initial template of how to organize patterns and their relationships. To that framework the notion of smells has been incorporated.

Rising’s format is extended slightly further by breaking patterns into 3 pieces: Issues, Solutions, and the Pattern implementation, and then finally creating Smells for the Issues. This increases information richness of the linkages and is described in more detail in section 3.4.1 below.

Mäntylä (2003) shows that smells need to be categorized to be more usable, which has been added as well.
3.3 Method

Table 4 provides the steps of the research method. For each activity, the rationale is identified in the last column. The relationships between patterns, issues, solutions and smells is described in Section 3.4.1 below.

<table>
<thead>
<tr>
<th>No.</th>
<th>Activity</th>
<th>Rational</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Review papers in business modeling and concurrency research (especially survey papers)</td>
<td>Create a framework for categorizing concurrency issues and solutions based on the distributed computing literature and BPM</td>
</tr>
<tr>
<td>2</td>
<td>Capture patterns, issues, and smells (depending on paper) in database</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Break issues and solutions out of pattern descriptions</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Derive additional smells from issues (all issues should in some way be connected to a smell)</td>
<td>Transform clustered issues into guidelines using terminology more familiar to business process modelers (vs. computer science)</td>
</tr>
<tr>
<td>5</td>
<td>Link records (nodes) based on descriptions and comparing similar nodes</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Categorize smells</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>To create guidelines, generate report for each data smell by traversing database</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Re-word names and descriptions to improve understandability and regenerate report</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 Research method
3.4 Data Acquisition

As mentioned earlier, data concurrency issues have not been widely studied in BPM research, and there is also relatively little literature on “Smells”. However, just as the term “smell” can be applied to “Guidelines for finding problems”, there are a lot of other terms that can be used as well. The VIDE Consortium did a systematic survey of the literature up to 2007 on quality defects in model driven software (Rech et al. 2007), and reviewed approximately 950 papers. From this, they created a list of related topics that can be used as search criteria to find more issues:


The search was also expanded beyond VIDE’s. Any research area dealing with reliability involving people and/or data interacting concurrently might contain patterns that could be applied to BPM. Some that come to mind are: Distributed/Concurrent Computing (e.g. Telecommunications & Real-time Systems), Management Information Systems (MIS), Software Reliability, Software Refactoring, Service Oriented Architectures (SOA), Business Process Re-engineering (BPR), and Human Reliability research.

Finally, literature surveys were focused on, in order to simplify finding a representative sample based on other researchers review criteria, and to make use of the correlation work they had already done.
3.4.1 Node Types

As mentioned earlier, patterns were broken into 3 pieces: Issues, Solutions and the Pattern's Implementation. This allowed a richer set of linkages to be created between them. Smells were them connected to sets of issues, which may connect multiple solutions and patterns, as can be seen in Figure 2 Note Types and Relations

\[
\text{Pattern} \rightarrow \text{Issue} \rightarrow \text{Solution} \rightarrow \text{Pattern} \rightarrow \text{Issue} \rightarrow \text{Solution} \rightarrow \text{Pattern}
\]

This can be described textually as (a formal entity relationship diagram is provided later):

- A Node can be a Pattern, Smell, Issue, or Solution
- A Pattern has zero or more Issues. A Smell has one or more Issues.
- An Issue Has zero or more Solutions (or solution Patterns), and is Had-by zero or more Patterns or Smells
- A Solution Has zero or more Patterns and Issues
• Ordinarily a Pattern is made of Issues connected to Solutions, but here a Pattern becomes primarily a reusable algorithm (way of doing things).

• All nodes should be associated with Smells, either directly or indirectly, (otherwise they will never appear in a guideline). Indirectly connected, means that a path exists from an Issue to a Smell, regardless of the direction of the links.

In addition, links can be drawn between nodes of the same type, as is shown in Figure 3 Integration Links.

• A Pattern may Use other Patterns and therefore inherit some of their Issues.

• Patterns, Smells, Issues and Solutions may be similar or opposite to other nodes of the same kind. These links are two-way relationships unlike all the previous types of links which are one-way.
Figure 4 shows the corresponding entity relationship diagram using Chen’s notation (Chen 1976). The double line between the Smell node and its Has relationship means that a Smell must be connected to at least one Issue directly. All the other nodes do not have any mandatory connections, although as mentioned before without a relationship connecting them indirectly to a Smell they will never appear in a guideline.

3.4.2 Database Structure

The actual database used to capture the nodes and links is a simple relational database,
with one table holding the Links, and another the Nodes, as shown in Figure 5. Figure 6 Database Entity Relation Diagram shows the corresponding entity relationship diagram (based on Riccardi 2002). With such a structure, it is possible to create any type of graph, as long as care is taken to keep the links and nodes in sync with each other. This structure allowed importing the data into various tools for editing, graphing, analysis and querying.

The report is generated by doing a depth first traversal of the tree rooted at each smell, on a section by section basis. This is discussed in more detail in Section 4.5 “Description Depth”.

In Figure 6, Link has double lines to show it is a weak entity which is identified by its To and From relationships. In addition Link must have To and From relationships (double line) while Node does not (single line). That is because a Link must be connected to nodes, while a Node does not need to be connected to any Links. Nodes use a unique

---

**Figure 5 Smell Database Structure**
number as a key. They also have a unique identifier but that is just an implementation
detail to be more humanly readable and not an important part of the framework.

Nodes contain references back to their sources so the modeler can find detailed
information how (or even if) the smell, issue, solution or pattern implementation applies
to their context.

When creating the database the information from each paper was captured before it was
connected to other sources. When a smell, issue, solution or pattern appears in more than
one source then it may be captured as two nodes connected to each other as being similar.
If a node truly is a duplicate of another, then it may still make sense to combine them to
reduce complexity, but their links to other nodes from their respective sources would still
remain.
For example: if many solution nodes had the same issue about not being supported in a particular language, it may make sense to combine them into a single “Not supported in language XYZ” issue node. The originating solution nodes will still have references back to where this information came from, which the reader could use to find a possible workaround from one (or more) of the related solutions’ documents.

Figure 4 showing the relationships between the node types, and Figure 6 showing the database attributes, can be combined together. The node types in Figure 4 are sub-classes of the Node entity in Figure 6, and the Link types in Figure 4 are sub-classed and simplified from the Link entity and To and From relations in Figure 6. They are diagramed separately to make relationships easier to understand.
4 RESULTS

This chapter is organized in six sections. Section 4.1 lists the specific sources used to create the database. Section 4.2 categorizes the pattern information. Section 4.3 discusses an orthogonal break-down into sections. Section 4.4 presents a detailed data analysis. Section 4.5 shows how different search depths affect the results, followed by section 4.6 which gives example Smell descriptions and guidelines. Finally, section 4.7 suggests an interactive approach to navigate the relationships between the guidelines.

4.1 Sources

Currently, the database contains only a small sampling of the types of papers discussed in “Data Acquisition” (Section 3.4 above). In the guidelines generated from the database (in the appendices) the papers will be referenced using these acronyms:

- **CBP** Concurrent Bug Patterns and How to Test Them, (Farchi et al. 2003)

- **DAP** Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows, (Trčka et al. 2009)

- **PAP** Process anti-patterns: How to avoid the common traps of business process modeling, (Koehler & Vanhatalo, 2007)

- **PMG** Seven Process Modeling Guidelines (7PMG), (Mendling et al. 2010)

- **WCP** Workflow Control-Flow Patterns: A Revised View, (Russell et al. 2006)

- **WDP** Workflow Data Patterns, (Russell et al. 2004a)
There are also a few records whose source is listed as "tbd". These are based on the author’s experience and supporting sources have not been entered into the database yet. In some cases where there is a source but it has not been fully captured in the database, it is documented in the description.

With the exception of DAP and PMG (which are fairly new) all of these survey papers are heavily referenced in the literature and therefore make a strong foundation for this exercise.

WDP and DAP both comment about the lack of research on data flow problems in Business Process Modeling. With the exception of CBP, this small set of papers may make up the majority of research in our specific research area. However, as discussed previously, there is a lot of related research into both data and concurrency that can be applied, CBP being just an example.

DAP, PMG, WCP, WDP and WCP, all have W.M.P. van der Aalst as one of the co-authors for these papers and can be considered as loosely related as far as content and supporting data.

4.2 Categories

The database re-uses the categories from the WorkflowPattern repository (WCP, WDP, and WRP):
• **Control**
  
The control-flow perspective captures aspects related to control-flow dependencies between various tasks (e.g. parallelism, choice, synchronization etc)

• **Data**
  
The data perspective deals with the passing of information, scoping of variables, etc.

• **Resource**
  
The resource perspective deals with resource to task allocation, delegation, etc.

The WorkflowPattern repository uses a fourth category as well, **Terminations**, which is applicable to Data Concurrency issues, but it was organized differently and not analyzed as part of this study. Also, one of the sections, which spanned the Control, Data, and Resource categories, deals specifically with Termination patterns.

**4.3 Sections**

Sections capture themes which may appear in one or more of the categories above. Many of these began as sub-categories from the WorkflowPattern repository but quickly expanded outside their original categories as information from the various sources was analyzed.

• **Concurrency**: Dealing with parallel execution
• **External**: Data interactions with entities external to the BPM. (originally only under Data in WDP)

• **Internal**: Data interactions internal to the BPM (originally just WDP)

• **Robustness**: Dealing with fault tolerance or avoidance

• **Routing**: How data can influence the behaviour BPM. (originally just WDP)

• **State**: Dealing with condition or mode of being, as in a Finite State Machine (originally just WCP)

• **Termination**: Dealing with the end of execution, intended or otherwise (originally just WCP)

• **Transfer**: How data may be transferred (originally just WDP)

• **Trigger**: External signals which affect control flow (originally just WCP)

• **Usability**: The comprehensibility of the model (primarily from PMG and PAP)

• **Usage**: The construction/syntax of the model

• **Visibility**: How data can be accessed (originally just WDP)

An additional section covered specific **Language** topics, but was not included in the analysis, because this thesis is intended to be independent of modeling languages, and since languages evolve these issues would have had to be checked for relevancy.
In WDP (Russell et al. 2004a), the External and Internal “sub-categories” are two halves of a larger sub-category dealing with Data Interactions. This is very closely related to Data Transfer which describes “how” External and Internal transfers operate.

Usability deals primarily with Comprehensibility, i.e. how easy is the model to understand by the author and others, especially when it comes to ensuring the model does what the author intended. Likewise, Testability, Maintainability, Complexity and other ‘ilities could also have been used. Usability is not the perfect name for this section, but is perhaps the simplest.

The Concurrency, Robustness, Usage, (and Language) sections were derived as part of the research for this thesis rather than being re-used from a specific source.

4.4 Data analysis

As discussed in section 4.1 above, the database is based on most of the available BPM data concurrency research as well as a few additional sources.

The database contains: 387 Nodes (i.e. records, 157 Data, 150 Control, and 80 Resource) as well as 679 Links: (506 Has, 69 Uses, 101 Similar, and 3 Opposite). It makes sense that it has more links than nodes. Most nodes were created by splitting pattern descriptions apart, and data smells were connected to issues. At a minimum the number of links cannot be much less than the number of nodes. A pattern description may have multiple issues, and implementations, as well as references to other patterns. Finally as part of the contributions of this research, connections were created between related nodes resulting in 75% more links than nodes.
Table 5 Database statistics

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>65</td>
<td>45</td>
<td>1</td>
<td>39</td>
<td>150</td>
</tr>
<tr>
<td>Data</td>
<td>56</td>
<td>41</td>
<td>25</td>
<td>35</td>
<td>157</td>
</tr>
<tr>
<td>Resource</td>
<td>19</td>
<td>43</td>
<td>0</td>
<td>18</td>
<td>80</td>
</tr>
<tr>
<td>Total</td>
<td>140</td>
<td>129</td>
<td>26</td>
<td>92</td>
<td>387</td>
</tr>
</tbody>
</table>

Figure 7 Graphical Representation of Database

4.5 Description Depth

The database is essentially a collection of directed graphs. Depending on the research source, each graph may be rooted in any type of node. As discussed previously, all Data Issues have been connected to at least one Smell, and links have been added between nodes creating new relationships when possible.

To create the descriptions of Data related Smells in the form of patlets, each Smell is essentially “picked” up and what comes with it become the issues and solutions within its guidelines. However, these connected graphs may contain dozens of nodes, with cyclic links so there has to be a stopping condition or some of the descriptions will be too large.
to be understandable. (Later, in section 4.7, an interactive approach to navigation is suggested to solve this problem)

Figure 8 Depth of 1 - Just Smell Nodes

Figure 8 shows a tree depth of one, i.e. just the Smell nodes (other nodes have been darkened to represent that they are not linked). See Appendix A for the list. A depth of two would include the Smells and their directly connected Issues.

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>13</td>
<td>10</td>
<td>1</td>
<td>20</td>
<td>44</td>
</tr>
<tr>
<td>Data</td>
<td>52</td>
<td>28</td>
<td>25</td>
<td>31</td>
<td>136</td>
</tr>
<tr>
<td>Resource</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>65</td>
<td>38</td>
<td>26</td>
<td>51</td>
<td>180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>20%</td>
<td>22%</td>
<td>100%</td>
<td>51%</td>
<td>29%</td>
</tr>
<tr>
<td>Data</td>
<td>93%</td>
<td>68%</td>
<td>100%</td>
<td>89%</td>
<td>87%</td>
</tr>
<tr>
<td>Resource</td>
<td>0%</td>
<td>0%</td>
<td>--</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Total</td>
<td>46%</td>
<td>29%</td>
<td>100%</td>
<td>55%</td>
<td>47%</td>
</tr>
</tbody>
</table>

Table 6 Database statistics for depth of 3

Table 6 shows the quantity and percentage of the total (from Table 5) for each type and category of node that has been “picked” up by traversing a tree to the depth of three. The
trees now fan out to all the nodes the Smells’ Issues are connected to, and covers most of the Data nodes. However, it only goes one level past the Issues. This will reach Patterns and Solutions in the tree, but none of the Uses links which are primarily between Patterns, and therefore the next level down.

This is shown graphically in Figure 9. Nodes that have been picked up as part of trees with a depth of three are shown in medium grey. Smell nodes, which are the roots of the trees, are still white.

Figure 10 shows a depth of four which does a much better job of capturing the interconnections.
Figure 10 Depth of 4

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>26</td>
<td>24</td>
<td>1</td>
<td>26</td>
<td>77</td>
</tr>
<tr>
<td>Data</td>
<td>54</td>
<td>39</td>
<td>25</td>
<td>34</td>
<td>152</td>
</tr>
<tr>
<td>Resource</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>81</td>
<td>65</td>
<td>26</td>
<td>61</td>
<td>233</td>
</tr>
</tbody>
</table>

Table 7 Statistics for depth of 4

Figure 11 continues to a depth of 5 and captures the last of Data’s Issues and Solutions. This also further increases the size of each Smell’s tree of related nodes, making them confusing and less useful.

Note, there is still one Data issue missing: “Data-based trigger not directly supported?” (WDP-39). It is connected to the “External” smell “Sending data outside of the Workflow” (WDP-15), indirectly as a form of delivery acknowledgement, but is at a depth of six. Strictly speaking it is one of the “Language” specific issues which were
excluded from the analysis, however since it has a workaround, it was placed in the “Routing” section instead. It also not a data concurrency issue.

![Figure 11 Depth of 5](image)

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>39</td>
<td>36</td>
<td>1</td>
<td>32</td>
<td>108</td>
</tr>
<tr>
<td>Data</td>
<td>55</td>
<td>41</td>
<td>25</td>
<td>35</td>
<td>156</td>
</tr>
<tr>
<td>Resource</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>Total</td>
<td>97</td>
<td>83</td>
<td>26</td>
<td>70</td>
<td>276</td>
</tr>
</tbody>
</table>

Table 8 Statistics for tree depth of 5

Nodes which are similar to each other can be considered to share all the same links, but since they are stored as a graph, shared links between similar nodes are treated as being one link further away. The amount of information can be increased while reducing the depth by flattening these “similarity” trees and treating them all as being at an equivalent level. Table 9 and Table 10 below, show the new statistics.
Table 9 Statistics for depth of 3 with similar nodes promoted.

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>17</td>
<td>31</td>
<td>1</td>
<td>23</td>
<td>72</td>
<td>26</td>
<td>69</td>
<td>100</td>
<td>59</td>
<td>48</td>
</tr>
<tr>
<td>Data</td>
<td>52</td>
<td>36</td>
<td>25</td>
<td>33</td>
<td>146</td>
<td>93</td>
<td>88</td>
<td>100</td>
<td>94</td>
<td>93</td>
</tr>
<tr>
<td>Resource</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>--</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>70</td>
<td>70</td>
<td>26</td>
<td>57</td>
<td>223</td>
<td>50</td>
<td>54</td>
<td>100</td>
<td>62</td>
<td>58</td>
</tr>
</tbody>
</table>

Table 10 Statistics for depth of 4 with similar nodes promoted

<table>
<thead>
<tr>
<th></th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
<th>Issue</th>
<th>Pattern</th>
<th>Smell</th>
<th>Solution</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>45</td>
<td>39</td>
<td>1</td>
<td>29</td>
<td>114</td>
<td>69</td>
<td>87</td>
<td>100</td>
<td>74</td>
<td>76</td>
</tr>
<tr>
<td>Data</td>
<td>54</td>
<td>41</td>
<td>25</td>
<td>34</td>
<td>154</td>
<td>96</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>98</td>
</tr>
<tr>
<td>Resource</td>
<td>3</td>
<td>9</td>
<td>0</td>
<td>4</td>
<td>16</td>
<td>16</td>
<td>21</td>
<td>--</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>102</td>
<td>89</td>
<td>26</td>
<td>67</td>
<td>284</td>
<td>73</td>
<td>69</td>
<td>100</td>
<td>73</td>
<td>73</td>
</tr>
</tbody>
</table>

Data node coverage increased noticeably (+6%) for a depth of 3, and slightly (+1%) for a depth of 4. However, the overall coverage of a tree with a depth of 3 and similar nodes promoted is very close to the overall coverage of a tree with the depth of 4 without promotion (58% vs. 60%). The over-all coverage of a tree with a depth of 4 with promotion is actually slightly better than the depth of 5 without promotion (73% vs. 71%), making this a notable improvement.

4.6 Example Smell Description

This section shows two examples, the first demonstrating expanding a description to a depth of 3, and the second to a depth of 4. Both promote similar nodes to be treated as siblings.
4.6.1 Example smell description, depth of 3

In Figure 12 below, a depth of 1 only includes the Smell “Making changes...”, while a depth of 2 captures the Issue “Passing data...”. It will also promote all similar issues, like “Unknowable outcome” regardless the length of similar links.

In this example there is only one similar link and “Unknowable outcome...” also happens to be directly connected to “Making changes...” (link not shown in diagram). When this happens, “Unknowable outcome...” should not show up in the Smell’s listing twice.

When expanding the depth of the description, the direction of the links is not important. In this example, at a depth of 3, only the links connected to “Passing data...” are followed. The Issue has the Solution “Merge changes...” and has a reverse connection to the Pattern “Task to Task”. A reverse connection means that it is the Pattern “Task to Task” that has the Issue “Passing data...”
In the example smell description listing below, as in the appendices, smells are divided into sections. Each individual smell, pattern, issue or solution’s name is followed by its category and its source. In the description for “Making changes…”, you can see it is in the Concurrency section, a member of the Data category, Concurrency section (of course), and its source is WDP (Russell et al. 2004a) pattern 29.

<table>
<thead>
<tr>
<th>Name</th>
<th>Cat.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Making changes to copied data</td>
<td>Data</td>
<td>Concurrency</td>
<td>WDP-29</td>
</tr>
</tbody>
</table>

Whenever data is shared by making copies of it, then it is very easy for the copies to cease being the same.

Consider Issue:

Passing data when control flow and data travel together

(also known as: Passing data when control and data channels are integrated)

If the data for an activity and control of the activity are integrated, and if the data was copied to parallel paths, then it is possible that two (or more) different copies of the data will arrive at the same location. For example, while building a house, copies of the plans are passed along to each new trade as they take over an area - along with notes, changes, deletions and possibly mistakes. Eventually someone will receive two copies of the plans which are no longer the same.
Consider Solution:
Merge changes strategy  Data Transfer  WDP-29

Related to Pattern:
Task to task  Data Internal  WDP-9

Unknowable outcome when copied data is updated
[...description...]

In the description listing, using “Has a” or “Had by” to express relations is not easy to understand, so “Has” links become “Consider” and reverse links (“Had by”) becomes “Related to”. The “similar” links have disappeared because they were promoted. So, in this example, you could read it as: If a practitioner sees a model where changes are being made to copied data (smell), then they should look for problems with passing data when control flow and data travel together (issue) and then consider using a merge changes strategy (solution) to fix it. They could also refer to the “Task to Task” pattern for more information.

You may notice that the smell “Making changes...” and solution “Merge change...” come from a different WDP source pattern than the issue “Passing data...” and pattern “Task to Task”. This relationship may not be obvious from the original source. This is even less likely if they came from different papers. The complete list (to date) of smell descriptions, to a depth of 3, can be found in Appendix B.
4.6.2 Example smell description, depth of 4

At the depth of 4, “uses” relationships finally appear. Unlike “has” relationships where in the textual description the word is replaced completely, “uses” is less difficult to understand, so “used by” is used to represent the reverse relationship.

![Figure 13 Example Tree - Depth 4](image)

In the example, the issue “Unknowable outcome” appears for the second time, this time directly linked (in reverse) to the solution “Merge changes”. Since it is in a different part of the description it needs to be left there. However, if it was at a higher level and was going to be expanded for the second time, the message “PLEASE SEE earlier description” is included rather than duplicating everything. This is only done per section so readers do not have to search too far in the document.
Concurrency:

Making changes to copied data

Data  Concurrency  WDP-29

[...description...]

Consider Issue:

Passing data when control flow and data travel together

Data  Internal  WDP-9

[...description...].

Consider Solution:

Merge changes strategy

Data  Transfer  WDP-29

Consider Issue:

Data overload

Data  Internal  WDP-11

Token backlog

Data  Robustness  tbd

Related to Issue:

Unknowable outcome when copied data is updated

Data  Transfer  WDP-29

Merging results

Data  Internal  WDP-12

Related to Pattern:

Task to task

Data  Internal  WDP-9

Consider Issue:

Common datastore concurrent access

Data  Concurrency  WDP-5

Reallocating a task needing contextual data

Data  Concurrency  WRP-30
Random problems                  Control Concurrency WCP-1
Data is late                      Data Internal WDP-9

**Uses Pattern:**

Task data

Unknowable outcome when copied data is updated

[...description...]

Note that the source of “Token backlog” is “tbd”. The issue comes from Börger et al. (2008) which has not been fully captured in the database. This issue’s description can be found in the Robustness section under “Using the token design pattern” (appendix section B.6.1).

In this example, you could expand the description for the “Making changes to copied data” smell as: *If you use the merge change strategy solution, then you should be aware that it has issues with data overload and token backlog. This is also a solution for issues with unknowable outcomes when copied data is updated and merging results. Similarly, if you choose to consider the "Task to task" pattern, then you should know that it has several issues as well to consider and it in turn uses the "Task data" pattern.*

### 4.7 Navigation

Description tree depths of four (with similar nodes promoted), do an acceptable job of covering the data concurrency guidelines. However, while a tree of depth 4 has approximately only 25% more nodes than a tree of depth 3 (see Table 9 and Table 10
above) these nodes are repeated often because of how tightly interlinked the nodes are (see Section 4.4). Promoting similar nodes further increases the number of children for each node. All together this can make it easy to lose track of where you are in the guidelines. To illustrate this, the last smell in appendix B: B.12.1 “Reusing names for data” takes over two pages as a guideline when expanded to the depth of three. When expanded to a depth of four it takes six pages. Breadcrumbs in the appendices headers help the reader keep track of which smell they are in, but it may still be hard for the reader to remember how the nodes on the fifth page are related to the nodes on the second.

A better way to handle this is with an interactive interface, not possible in a printed document. There are many choices of data browsers, both textual and graphical. For purposes of an example a simple wiki interface was created. Wikis have many advantages for sharing data on the web, and allowing other researchers to add and review content (Weiss et al. 2007b).

In Figure 14 below, a wiki based on Tiddlywiki (specifically Shulman 2009) was created using a similar report generator to the one that created the appendices at the end of the thesis.

The figure shows the same smell guideline as discussed section 4.6.1, but only with only selected branches expanded. The top of the complete list of smells is shown as well. While not shown due to limitations on the size of the printed page, it is possible to drill as deep as desired, including following loops. While this can easily produce too much
information it is easy to see where you are in the tree, and to collapse parts of the tree that are not of immediate interest. It is also possible to have multiple browsers and use different search criteria to create different views of the guidelines.
This was just a simple example, and is left as a suggestion for future research with regards to its effectiveness.
5 CONCLUSIONS, LIMITATIONS, AND FUTURE RESEARCH

5.1 Conclusions

Business processes suffer from the same problems as distributed computer systems, even when the work is being done by humans. While there is a great deal of literature on concurrency issues in distributed systems, parallelism and related disciplines in computer science, research in the area of business process modeling is still young and focuses primarily on control flow problems.

The objective of this research was to move modeling knowledge from computer science to the business domain and thus make it available to business process modelers. As there is a gap in the business process modeling research on data concurrency, the thesis specifically focuses on creating guidelines for where to look in models for data concurrency problems, what those problems may be, and possible solutions.

5.2 Key findings

1 Patterns are typically organized by the types of solutions they provide and then refer to the contexts to which they may be applied.

   a. Reorganizing patterns into smells provided guidelines that showed the reader how to recognize situations which may hide errors, and then lists what the problems may be, how to solve them, and what other issues and problems may present. This is a novel way to view information about data concurrency problems.
b. The process of codifying the data concurrency knowledge broke the patterns and other knowledge from different sources into their component parts and re-connected the related parts again. While previous research documented related patterns, this process synthesized a richer coupling by combining others’ research. This is a generic process that can be applied to other domains, and similarly help extract knowledge from patterns and principles (guidelines) documented in the literature.

2 Literature on data concurrency issues in modeling is typically written in terms not familiar to business people. The language of computer science, with regards to data concurrency, was adapted to language more familiar to the business analysts who design business processes. These proof-of-concept guidelines will help BPM practitioners improve the quality of their models, by being able to make use of knowledge about data concurrency problems, well known in the realm of computer science, of which they were previously unaware.

5.3 Limitations

The study has three limitations:

1. The database is incomplete. Only seven out of over a hundred papers and books containing collections of data concurrency problems have been captured in the database.

2. As the current database is small, the framework will need to evolve in order to scale with the complexity of categorizing larger numbers of patterns and being able to use the resulting multitude of guidelines.
3 Categorization was done manually by a single researcher.
4 Usefulness of guidelines has not been validated with domain experts.

5.4 Opportunities for future research

The current research opens many possibilities for future research:

1 Expand the BPM pattern database with additional concurrency patterns
   a. Especially in control flow and resource management
   b. It would be very interesting to extend this into related performance, fault
tolerance and security areas. (Bonabeau 2002)

2 Validate guidelines with subject matter experts to discover if the guidelines are
   useful and understandable. An “Expert panel” approach is commonly used in
   Medicine to validate symptom diagnoses, and only requires a small number of
   experts. Goldenberg et al. (2001) provide an example of it applied to business
   research.

3 A dynamic interface to navigate the Smells, Issues, and Solutions would make it
   much easier for readers understand the interrelations than the static listings in the
   appendices below. A database browser or a custom Wiki would be immensely
   helpful (Weiss et al. 2007b).

4 Experiment with categorization algorithms to see if there is a more effective
   grouping.
   a. Different types of links may be considered to be of different value
   b. Current manual groupings carry relationship information as well
c. Using coupling and cohesion metrics to form the groups (which may include
groups within groups) based on connection weights may be better than the
 technique of simply breaking links to form groups.

5 Automated analysis may be more effective.

a. Would require all the patterns, issues, guidelines, etc. from the various sources
 be translated into a consistent formal syntax first.

b. Keyword analysis may be useful but even papers by the same author do not
 always use words consistently.

6 The list of data concurrency problems could also be used by BPM tool designers to
 improve model rule checkers.

7 Apply framework to other pattern domains.
A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared

6 REFERENCES


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared


A.1 Concurrency:
A.1.1 Using a lock when making changes to something shared

A. DATA CONCURRENCY GUIDELINE LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1.1 Using a lock when making changes to something shared</td>
<td>Data</td>
<td>Concurrency</td>
<td>CBP-2.2</td>
</tr>
</tbody>
</table>

A standard way to avoid concurrency problems is to prevent more than one process or person from making changes to shared data, or access a shared resource, is to lock access to it - like a shared bathroom, until it is available for others to use. This can result in other issues.

A.1.2 Making changes to copied data | Data | Concurrency | WDP-29 |

Whenever data is shared by making copies of it, then it is very easy for the copies to cease being the same.

A.1.3 Concurrency management | Data | Concurrency | WDP-29 |

Different modeling languages have many ways to manage local and remote concurrent access to data and resources, including doing nothing.

A.1.4 Accessing shared data | Data | Concurrency | WDP-5 |

A.2 External:

A.2.1 Sending data outside of the workflow | Data | External | WDP-15 |

Sending data into the environment (outside the Workflow/BPM)

A.2.2 Receiving data or requests from outside workflow | Data | External | WDP-17 |

The outside environment can push data into the workflow at any time. Similarly, when the outside environment may ask at any time for information from the
A.3 Internal:
A.2.3 Retrieving data from outside of the workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflow</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A.2.3 Retrieving data from outside of the workflow

Data External WDP-16

The workflow needs to retrieve information from the outside world.

A.3 Internal:

A.3.1 Data used by multiple instances

Data Internal WDP-1

Multiple instances of a process use the same data, for example multiple releases of a product all using the same installation manual.

A.3.2 Separate data and control flow channels

Data Internal WDP-9

Control and data travel separately to tasks. For example tasks are triggered by a schedule but the data that needs to be processed is sent electronically (or through the mail, etc.)

A.3.3 Merging results from multiple instance tasks

Data Internal WDP-12

When multiple instances of a sub-process have finished does their data need to be merged back together in order to continue?

A.3.4 Shared case data

Data Internal WDP-14

Sharing data between two related instances of a workflow (i.e. separate but related cases)

A.3.5 Bypassing gateways

Data Internal PAP-5.6

Data bypasses gateways because it is not needed until some later point in the execution
A.4 Robustness:

A.3.6 Dangling inputs and outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3.6 Dangling inputs and outputs</td>
<td>Data</td>
<td>Internal</td>
<td>PAP-5.1</td>
</tr>
</tbody>
</table>

Inputs and outputs of an activity or gateway that remain unconnected in the model, very often as residues of connections that users decide to delete or redirect. (Many BPM tools/languages will catch this automatically, but not all)

A.4 Robustness:

A.4.1 Existence check

Checking if a value or object currently exists.

A.5 Routing:

A.5.1 Data-based routing

When the evaluation of data-based expressions have the ability to alter the control-flow within a case.

A.6 State:

A.6.1 Using the token design pattern

A token is an exclusively held resource that is analogous to a physical object. Only one party can hold a token at any given time and if the token is destroyed, it cannot be used at all. This pattern is crucial to how many modeling languages work (e.g. BPMN), however the pattern can be used within a Business Processes as well, for example: in formal meetings only the person who "has the floor" is supposed to speak.
A.7 Termination:
A.7.1 Multiple termination points

A.7 Termination:

A.7.1 Multiple termination points

In some modeling languages there are stop nodes which shutdown the whole process as well as end nodes which just terminate an execution branch. The stop nodes may also be needed to release data from a sub-process to its parent process when it finishes. Therefore a model might contain stop/shutdown nodes on multiple branches.

A.7.2 Not cleaning up data

Data is not cleaned up after use, or ever

A.8 Transfer:

A.8.1 Passing data to a different data type

The Sender and Receiver use different data types when passing information

A.9 Trigger:

A.9.1 Events as control flow

Frequently control flow is used to capture events.

A.10 Usability:

A.10.1 Multiple connections between activities

Complex control and data flows easily lead to multiple connections in process models, which can result in cluttered models.
A.11 Usage:
A.10.2 Cluttered data-flow models

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.10.2 Cluttered data-flow models</td>
<td>Data</td>
<td>Usability</td>
<td>PAP-5.2</td>
</tr>
</tbody>
</table>

A.11 Usage:

A.11.1 Data is written to, but never used
Data Usage DAP-2

A.11.2 Data is over-written without being used first
Data Usage DAP-4

A.11.3 Data is read or destroyed before being initialized or written to.
Data Usage DAP-1

A.12 Visibility:

A.12.1 Reusing names for data
Data Visibility WDP-1

Reusing names for data elements in different parts of the process model.
B.1 Concurrency:
B.1.1 Using a lock when making changes to something shared

**B. DATA CONCURRENCY GUIDELINES AND ISSUES (DEPTH OF 3)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Concurrency:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.1.1 Using a lock when making changes to something shared</td>
<td>Data</td>
<td>Concurrency</td>
<td>CBP-2.2</td>
</tr>
</tbody>
</table>

A standard way to avoid concurrency problems is to prevent more than one process or person from making changes to shared data, or access a shared resource, is to lock access to it - like a shared bathroom, until it is available for others to use. This can result in other issues.

**Consider Issue:**

**Object assumed to be protected**

(Also known as: Code assumed to be protected)

Assuming access by others is prevented when it actually isn't (like people who assume a bathroom door is always locked when the bathroom is in use)

**Wrong lock or no lock**

If access to something is only controlled by an agreed upon "protocol" that has no way to absolutely control that access, then problems may occur when the protocol is not followed correctly or at all. For example people who are used to locked bathroom doors showing occupancy vs people who expect others to knock before opening the door.

**Non-protected version is used instead of protected version**

If not all the ways to access something are protected then using the unprotected access (unnecessarily) may cause problems. For example always accessing a computer through an administrator (or root) account can cause a lot of problems.
B.1 Concurrency:
B.1.1 Using a lock when making changes to something shared

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protection and subclassing</td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-4.1</td>
</tr>
<tr>
<td><em>(also known as: Confinement and Subclassing)</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-checked locking</td>
<td>Data</td>
<td>Creation</td>
<td>2.2.4</td>
</tr>
<tr>
<td>This comes from software, where a newly created object is available for others to use before it has been completely initialized. Similarly a report could be released for others to use while the author is still gathering information.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Nonatomic operations assumed to be atomic</strong></td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-2.2.1</td>
</tr>
<tr>
<td>Sometimes an activity that looks like it cannot be interrupted (self contained, i.e. atomic) can be because of the underlying activities which implement it. For example asking one person to sign a report not realizing they are really the contact person into an organization with its own approval process.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consider Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical section</td>
<td>Control</td>
<td>State</td>
<td>WCP-39</td>
</tr>
<tr>
<td>Interleaved routing</td>
<td>Control</td>
<td>State</td>
<td>WCP-40</td>
</tr>
<tr>
<td>Transfer by reference - with lock</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-31</td>
</tr>
<tr>
<td><strong>Two-stage access bug pattern</strong></td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-2.2.2</td>
</tr>
<tr>
<td>Sometimes a sequence of operations needs to be protected but instead it was wrongly assumed that protecting each operation separately is enough. For example transferring money electronically from one bank to another. It is not desirable if at any time that person appeared to have twice the amount of money or none.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transfer is not atomic.</strong></td>
<td>Data</td>
<td>Transfer</td>
<td>CBP-4.2</td>
</tr>
<tr>
<td>If control of something depends having possession of an object (for example a baton, or a crown) but the object is only exists digitally (e.g. a token), then it is possible for that object to exist in more than one place or cease to exist</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.1 Concurrency:
B.1.1 Using a lock when making changes to something shared

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clean-up after transfer</td>
<td>Data</td>
<td>Transfer</td>
<td>CBP-4.2</td>
</tr>
<tr>
<td>The token design pattern</td>
<td>Data</td>
<td>State</td>
<td>CBP-4.2</td>
</tr>
<tr>
<td>Using the token design pattern</td>
<td>Data</td>
<td>State</td>
<td>CBP-4.2</td>
</tr>
</tbody>
</table>

Contention problems with shared resources

Components of a process may find it difficult to function independently of data changes and concurrency constraints that occur in the broader process environment. For example: other components make changes to shared data that impacts this component (which needs the data to be stable), or other components lock access to shared data preventing this component from making the changes it needs.

Consider Pattern:
Transfer - copy in/copy out  Data  Transfer  WDP-29
Transfer by value - outgoing  Data  Transfer  WDP-28
Transfer by value - incoming  Data  Transfer  WDP-27

Consider Solution:
Version control  Data  Visibility  tbd

Related to Smell:
Concurrency management  Data  Concurrency  WDP-29
B.1 Concurrency:

B.1.2 Making changes to copied data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inconsistent data</td>
<td>Data</td>
<td>Usage</td>
<td>DAP-6</td>
</tr>
</tbody>
</table>

Data is inconsistent if a task is using this data while some other task (or another instance of the same task) is writing to this data or is destroying it in parallel.

B.1.2 Making changes to copied data

Whenever data is shared by making copies of it, then it is very easy for the copies to cease being the same.

Consider Issue:

Passing data when control flow and data travel together

(Also known as: Passing data when control and data channels are integrated)

If the data for an activity and control of the activity are integrated, and if the data was copied to parallel paths, then it is possible that two (or more) different copies of the data will arrive at the same location. For example while building a house, copies of the plans are passed along to each new trade as they take over an area - along with notes, changes, deletions and possibly mistakes. Eventually someone will receive two copies of the plans which are no longer the same.

Consider Solution:

Merge changes strategy

Related to Pattern:

Task to task

Unknown outcome when copied data is updated

Difficulties can arise where data is passed to a sub-activity that executes independently of the original activity. If these activities continue in parallel, then there is no way to know for certain when the sub-activity will end and
B.1 Concurrency:
B.1.3 Concurrency management

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>will copy its data back to the original. If the data has changed then it is important to know what impact this will have on the original (since you don't know when it will happen).</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>Consider Solution:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No asynchronous sub-routine calls</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>Do not copy data back</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>Merge changes strategy</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>Related to Pattern:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer by value - outgoing</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-28</td>
</tr>
<tr>
<td>Transfer - copy in/copy out</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>Transfer by value - incoming</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-27</td>
</tr>
</tbody>
</table>

B.1.3 Concurrency management

Different modeling languages have many ways to manage local and remote concurrent access to data and resources, including doing nothing.

Consider Issue:

Common datastore concurrent access

Where a common data store is utilized by several tasks for communicating data elements (rather than communicating the data between them directly), there is the potential for concurrency problems to arise, particularly if the case involves parallel execution paths. This may lead to inconsistent results depending on the task execution sequence that is taken.

Consider Solution:

No asynchronous sub-routine calls
Language specific serialization

Related to Pattern:

Block data
**B.1 Concurrency:**

**B.1.3 Concurrency management**

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-5</td>
</tr>
<tr>
<td>Folder data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-6</td>
</tr>
<tr>
<td>Scope data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-3</td>
</tr>
<tr>
<td>Workflow data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-7</td>
</tr>
<tr>
<td>Task to task</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-9</td>
</tr>
<tr>
<td>Case to case</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-14</td>
</tr>
</tbody>
</table>

**Reallocating a task needing contextual data** Data Concurrency WRP-30

*(also known as: Stateful Reallocation Data Concurrency)*

When reallocating a task to another resource care must be taken not to lose contextual information, and that there are no issues arising from concurrent use during the transfer period. For example making sure all the documentation is in place before transferring a project to another department, as well as any updates made afterwards by the original owners are transferred to the new owners as well.

**Consider Pattern:**

- Stateless reallocation  
  Resource Detour WRP-31
- Delegation  
  Resource Detour WRP-27
- Stateful reallocation  
  Resource Detour WRP-30

**Consider Solution:**

- Stateful reallocation data concurrency  
  Resource Concurrency WRP-30

**Random problems**

Mismanaging concurrency can cause different results depending on what order activities happen. Issues may appear as seemingly random behaviour. Data may be lost or corrupted, but only on rare occasions. They can also cause performance problems, or create holes that larger problems may escape through unnoticed until much later.
B.1 Concurrency:
B.1.3 Concurrency management

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple tokens</td>
<td>Control</td>
<td>Concurrency</td>
<td>WCP-1</td>
</tr>
<tr>
<td>Safe process model</td>
<td>Control</td>
<td>Concurrency</td>
<td>WCP-1</td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>Control</td>
<td>Basic</td>
<td>WCP-1</td>
</tr>
<tr>
<td>(also known as: Sequential routing, serial routing.)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Something tries to access results before ready**

For example someone not checking if a report is finished before making it public.

**Consider Solution:**

Use join to wait.  

**Related to Pattern:**

The fork/join design pattern  

The sleep bug pattern

Essentially assuming that one activity is faster than another so adding a delay to synchronize them, when in fact there is no guarantee one will be faster than the other (e.g. The fable of the Hare taking a nap because he thought he had plenty of time to win the race versus the Tortoise). Mostly it is a performance problem, but it can also cause race conditions with seemingly random results.

**Consider Solution:**

Use join to wait.

**Assume concurrency will never occur**

(also known as: Interleavings assumed never to occur)
B.1 Concurrency:
B.1.4 Accessing shared data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>The assumption that certain activities will never happen in parallel so there is no need to prevent them (e.g. two people updating an on-line document at the same time).</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>Losing a notify bug pattern</td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-2.3.2</td>
</tr>
<tr>
<td>When depending on messages to control access to a shared object, and not everyone is listening when the message is sent.</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-10</td>
</tr>
<tr>
<td>Contention problems with shared resources</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td>PLEASE SEE earlier description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subprocesses can access everything</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-10</td>
</tr>
<tr>
<td>Information is accessible by everything as soon as it is available, so data is passed to other tasks simply by being made available, hence concurrent tasks may try to make changes to the same data at unexpected times.</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-2</td>
</tr>
<tr>
<td>Consider Pattern:</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-2</td>
</tr>
<tr>
<td>Block data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related to Smell:</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-1</td>
</tr>
<tr>
<td>Reusing names for data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related to Solution:</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-10</td>
</tr>
<tr>
<td>Implicit data passing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.1.4 Accessing shared data</td>
<td>Data</td>
<td>Concurrency</td>
<td>WDP-5</td>
</tr>
<tr>
<td>Consider Issue:</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-5</td>
</tr>
<tr>
<td>Case data access</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

One consideration that arises with the use of case level data is in ensuring that these elements are accessible, where required, to the components of a subprocess associated with a specific case (e.g. as the definition of a block task).
B.1 Concurrency:
B.1.4 Accessing shared data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explicit case data access</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-5</td>
</tr>
<tr>
<td>Implicit case data access</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-5</td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-5</td>
</tr>
<tr>
<td><strong>Multiple instance data availability</strong></td>
<td>Data</td>
<td>Internal</td>
<td>WDP-13</td>
</tr>
<tr>
<td>One issue that arises with multiple instance tasks is identifying the point at which the output data from them are available (in some aggregate form) to subsequent tasks.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>From multiple instance task</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-13</td>
</tr>
<tr>
<td><strong>Result not available after join</strong></td>
<td>Data</td>
<td>Visibility</td>
<td>CBP-4.3</td>
</tr>
<tr>
<td>When an activity has several tasks happening in parallel, changes may be made to data which is lost when the separate tasks join back together. For example, having a document updated by several people in parallel but not ensuring that all those changes have been made it into the final version.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consider Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer by reference - with lock</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-31</td>
</tr>
<tr>
<td>Critical section</td>
<td>Control</td>
<td>State</td>
<td>WCP-39</td>
</tr>
<tr>
<td>Interleaved routing</td>
<td>Control</td>
<td>State</td>
<td>WCP-40</td>
</tr>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use higher scope of variables for task</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP</td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The fork/join design pattern</td>
<td>Control</td>
<td>Synchronizer</td>
<td>CBP-4.3</td>
</tr>
</tbody>
</table>

85
B.2 External:
B.2.1 Sending data outside of the workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**B.2 External:**

**B.2.1 Sending data outside of the workflow**

Sending data into the environment (outside the Workflow/BPM)

**Consider Issue:**

**Assuring data delivery**

When sending data to an external application how do you know if it was successfully delivered? This is particularly a problem when the external application does not provide immediate feedback on delivery status.

**Consider Solution:**

Delivery verification

Delivery acknowledgement

**Related to Pattern:**

Task to environment - push

Case to environment - push

Environment to case - pull

Environment to task - pull

Environment to workflow - pull

Workflow to environment - push

**Appropriate time to push**

Determining the proper time to push the data outside to the environment.

**Consider Solution:**

Rule-based communication scheduling
B.2 External:
B.2.2 Receiving data or requests from outside workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task to environment - push</td>
<td>Data</td>
<td>External</td>
<td>WDP-15</td>
</tr>
<tr>
<td>Case to environment - push</td>
<td>Data</td>
<td>External</td>
<td>WDP-19</td>
</tr>
<tr>
<td>Environment to case - pull</td>
<td>Data</td>
<td>External</td>
<td>WDP-20</td>
</tr>
<tr>
<td>Environment to task - pull</td>
<td>Data</td>
<td>External</td>
<td>WDP-16</td>
</tr>
<tr>
<td>Environment to workflow - pull</td>
<td>Data</td>
<td>External</td>
<td>WDP-24</td>
</tr>
<tr>
<td>Workflow to environment - push</td>
<td>Data</td>
<td>External</td>
<td>WDP-23</td>
</tr>
</tbody>
</table>

**B.2.2 Receiving data or requests from outside workflow**

The outside environment can push data into the workflow at any time. Similarly, when the outside environment may ask at any time for information from the workflow.

**Consider Issue:**

**Data source identification**

Is there any way for an outside application to know who within the over-all workflow it needs to send information to (or ask for information from)

**Consider Solution:**

Inform environment

**Related to Pattern:**

Environment to task - push
Task to environment - pull
Case to environment - pull
Workflow to environment - pull
Environment to case - push
Environment to workflow - push

**Data**

**External**

**WDP-17**

**WDP-18**

**WDP-22**

**WDP-26**

**WDP-21**

**WDP-25**
B.2 External:
B.2.2 Receiving data or requests from outside workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need additional data path for returned info.</td>
<td>Data</td>
<td>External</td>
<td>WDP-16</td>
</tr>
</tbody>
</table>

If data can be sent to the workflow at any time, then the actual implementation of the process model needs some mechanism for the data to arrive via. Especially if it was something in the workflow that requested the information, but is not passively waiting for it (i.e. it continues to execute other things rather than stopping and waiting)

**Consider Solution:**
Asynchronous communication facilities  
Data  
External  
WDP-17

**Related to Solution:**
Don't wait for data response  
Data  
External  
WDP-16

**Wait and respond to data requests without impacting progress**  
Data  
External  
WDP-17

If data can be sent to the workflow at any time then the workflow needs a way to know it has arrived without impacting the workflow's performance (e.g. by stopping or continually checking for it.)

**Consider Solution:**
Asynchronous communication facilities  
Data  
External  
WDP-17

**Related to Pattern:**
Environment to task - push  
Data  
External  
WDP-17
Task to environment - pull  
Data  
External  
WDP-18
Case to environment - pull  
Data  
External  
WDP-22
Workflow to environment - pull  
Data  
External  
WDP-26
Environment to case - push  
Data  
External  
WDP-21
Environment to workflow - push  
Data  
External  
WDP-25

**Data notification**

During the course of the execution of a case, new data may become available
B.2 External:
B.2.3 Retrieving data from outside of the workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>or the values of existing items may change. A means of communicating these data updates to the appropriate case is required.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Consider Solution:**

- Data notification at start of case
- Provide mechanism to update case data

**Related to Pattern:**

- Environment to case - push
- Environment to workflow - push
- Case to environment - pull
- Workflow to environment - pull

B.2.3 Retrieving data from outside of the workflow

The workflow needs to retrieve information from the outside world.

**Consider Issue:**

- Request prevents progress

This style of interaction can block progress of the requesting case if the external application has a long delivery time for the required information or is temporarily unavailable.

**Consider Solution:**

- Don't wait for data response

**Related to Pattern:**

- Environment to task - pull
- Environment to case - pull
- Case to environment - push
- Task to environment - push
B.2 External:
B.2.3 Retrieving data from outside of the workflow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow to environment - push</td>
<td>Data</td>
<td>External</td>
<td>WDP-23</td>
</tr>
<tr>
<td>Environment to workflow - pull</td>
<td>Data</td>
<td>External</td>
<td>WDP-24</td>
</tr>
</tbody>
</table>

**Appropriate time to pull**

Another issue is determining the best time for both a particular case and the environment to pull in data from outside the workflow..

**Consider Solution:**

Beginning of task or case

**Related to Pattern:**

Environment to task - pull
Environment to case - pull
Case to environment - push
Task to environment - push
Workflow to environment - push
Environment to workflow - pull

**Requires explicit support for the extraction of data from outside**

The implementation of the workflow needs to actually be capable of extracting data from external applications.

**Consider Solution:**

Database mapping functions

**Related to Pattern:**

Environment to task - pull
Environment to case - pull
Case to environment - push
Task to environment - push
B.3 Internal:
B.3.1 Data used by multiple instances

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow to environment - push</td>
<td>Data</td>
<td>External</td>
<td>WDP-23</td>
</tr>
<tr>
<td>Environment to workflow - pull</td>
<td>Data</td>
<td>External</td>
<td>WDP-24</td>
</tr>
</tbody>
</table>

B.3 Internal:

B.3.1 **Data used by multiple instances**

Multiple instances of a process use the same data, for example multiple releases of a product all using the same installation manual.

**Consider Issue:**

*Multiple instance data initialization*

When a task is able to execute multiple times but not all instances are created at the same time, then are the values of data elements for all execution instances set when the multiple instance task is initiated or can they be set after this occurs for each invocation of a task instance.

**Related to Pattern:**

*Sequence*  
Control Basic WCP-1

*(also known as: Sequential routing, serial routing.)*

*Task data*  
Data Visibility WDP-1

*To multiple instance task*  
Data Internal WDP-12

B.3.2 **Separate data and control flow channels**

Control and data travel separately to tasks. For example tasks are triggered by a schedule but the data that needs to be processed is sent electronically (or through the mail, etc.)
B.3 Internal:
B.3.2 Separate data and control flow channels

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
</table>

**Consider Issue:**

**Data is late**

When data and control-flow occur via separate communication channels, then it is possible the control-flow reaches a task before the required data elements have been passed to it. For example, your utility payment is due, but the cheque is still in the mail.

**Consider Solution:**

Wait for data

Wait to be processed

**Related to Pattern:**

Task to task

**Related to Solution:**

Explicit data passing via data channels

**Bypassing synchronizing gateways**

Dataflow bypassing synchronization gateways of alternative branches can cause deadlocks because data needed by a future operation may not be created along some branch. Note: Bypassing forks and joins in parallel flows does not cause a deadlock, because all branches execute in parallel and thus, all business items always arrive through the connections i.e., all tasks receive their inputs as specified.

**Consider Solution:**

Make input optional

**Related to Smell:**

Bypassing gateways

**Related to Solution:**

Explicit data passing via data channels
B.3 Internal:
B.3.2 Separate data and control flow channels

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A thread of control never arrives (lost tokens)</td>
<td>Control</td>
<td>Robustness</td>
<td>WCP-3</td>
</tr>
</tbody>
</table>

The use of the Synchronization pattern can potentially give rise to a deadlock in the situation where one of the incoming branches fails to deliver a thread of control to the join construct.

**Consider Solution:**

Structured process (no interrupts/exceptions)                       | Control    | Usage      | WCP-3  |
Cancel region                                                        | Control    | Cancellation| WCP-29 |

**Related to Pattern:**

Synchronization (m-of-m and-join)                                    | Control    | Synchronizer| WCP-3  |
*(also known as: AND-join, rendezvous, synchronizer)*
Thread merge (and-join)                                             | Control    | Discriminator| WCP-41 |
Structured synchronizing merge (or-join)                             | Control    | Synchronizer| WCP-7   |
*(also known as: Synchronizing join, synchronizer, OR-join)*
Generalized and-join (m-of-m and-join)                              | Control    | Synchronizer| WCP-33  |
Local synchronizing merge (or-join)                                  | Control    | Synchronizer| WCP-37  |

**Detecting undefined vs empty/null**

Being able to differentiate between data elements that have an undefined value and those that are merely empty or null.

**Consider Solution:**

Deciding undefined vs empty/null                                     | Data       | Visibility | WDP-34 |

**Related to Pattern:**

Task precondition - data existence                                  | Data       | Routing    | WDP-34 |
Task postcondition - data existence                                 | Data       | Routing    | WDP-36 |
Task postcondition - data value                                     | Data       | Routing    | WDP-37 |
B.3 Internal:
B.3.2 Separate data and control flow channels

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task precondition - data value</td>
<td>Data</td>
<td>Routing</td>
<td>WDP-35</td>
</tr>
</tbody>
</table>

**Related to Smell:**

Existence check  
Data Robustness  
WDP-34

**Missing tokens**

Individual abended threads/tokens would prevent a Join from resetting and failure to receive input on all of the incoming branches may result in the process instance (and possibly other process instances) stalling. Missing tokens could be a consequence of: - a design error - or that one of the tasks in the branch failing to complete successfully (e.g. as a consequence of it experiencing some form of exception) - or because the thread of control is passed outside of the branch. - or there are preceeding splits or joins or they are un-structured in form (i.e. un-balanced splits and joins) (from WCP3)

**Consider Solution:**

Cancel region
Control Cancellation  
WCP-29

Cancelling partial join (n-of-m and-join)
Control Robustness  
WCP-30

**Related to Pattern:**

Structured partial join (n-of-m and-join)
Control Discriminator  
WCP-30

Structured discriminator (1-of-m and-join)
Control Discriminator  
WCP-9

*(also known as: 1-out-of-M join.)*

Static partial join for multiple instances (n:m &-join)
Control Multiple_instance  
WCP-34

Multiple instances with a priori run-time knowledge (m:m and-join)
Control Multiple_instance  
WCP-14

Multiple instances with a priori design-time knowledge (m:m and-join)
Control Multiple_instance  
WCP-13

Multiple instances without a priori run-time knowledge
Control Multiple_instance  
WCP-15
B.3 Internal:
B.3.2 Separate data and control flow channels

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic partial join for multiple instances</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-36</td>
</tr>
<tr>
<td>Blocking partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-31</td>
</tr>
<tr>
<td>Blocking discriminator (1-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-28</td>
</tr>
<tr>
<td>Cancelling partial join for multiple instances (n:m and-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-35</td>
</tr>
</tbody>
</table>

**Related to Smell:**

Using the token design pattern

<table>
<thead>
<tr>
<th>Discriminator missing tokens</th>
<th>Categ.</th>
<th>Robustness</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual abended threads/tokens would prevent a Discriminator from resetting and failure to receive input on all of the incoming branches may result in the process instance (and possibly other process instances) stalling.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Consider Pattern:**

Cancelling discriminator (1-of-m and-join)

| Cancelling partial join (n-of-m and-join)                    | Control      | Discriminator        | WCP-32 |

**Related to Pattern:**

Structured discriminator (1-of-m and-join)

(Also known as: 1-out-of-M join.)

| Structured partial join (n-of-m and-join)                    | Control      | Discriminator        | WCP-30 |
| Static partial join for multiple instances (n:m &-join)      | Control      | Multiple_instance    | WCP-34 |
| Multiple instances with a priori run-time knowledge (m:m and-join) | Control      | Multiple_instance    | WCP-14 |
| Multiple instances with a priori design-time knowledge (m:m and-join) | Control      | Multiple_instance    | WCP-13 |
B.3 Internal:
B.3.3 Merging results from multiple instance tasks

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple instances without a priori run-time knowledge</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-15</td>
</tr>
<tr>
<td>Dynamic partial join for multiple instances</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-36</td>
</tr>
<tr>
<td>Blocking discriminator (1-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-28</td>
</tr>
<tr>
<td>Blocking partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-31</td>
</tr>
<tr>
<td>Cancelling partial join for multiple instances (n:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-35</td>
</tr>
</tbody>
</table>

**Passing data to subprocesses**

The actual implementation of a process needs a way to make data available to its sub-processes

**Consider Solution:**

- Implicit data passing
  - Data
  - Internal
  - WDP-10
- Explicit data passing via parameters
  - Data
  - Internal
  - WDP-10
- Explicit data passing via data channels
  - Data
  - Internal
  - WDP-10

**Related to Pattern:**

- Block task to sub-workflow decomposition
  - Data
  - Internal
  - WDP-10
- Sub-workflow decomposition to block task
  - Data
  - Internal
  - WDP-11

**B.3.3 Merging results from multiple instance tasks**

When multiple instances of a sub-process have finished does their data need to be merged back together in order to continue?

**Consider Issue:**

- Merging results
  - Data
  - Internal
  - WDP-12

When instance specific data is passed by value (i.e. copies), then if at the
B.3 Internal:
B.3.4 Shared case data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>conclusion of the tasks the data needs to be reassembled then the data may not go back to together as easily as it came apart. For example after each contributor has updated their copy of the report, all the changes need to be merged back together.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared data passed by reference</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-12</td>
</tr>
<tr>
<td>Merge changes strategy</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-29</td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>From multiple instance task</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-13</td>
</tr>
</tbody>
</table>

B.3.4 Shared case data

Sharing data between two related instances of a workflow (i.e. separate but related cases)

**Consider Issue:**

**Linking shared case data**

Establishing an effective means of linking related cases and their associated data together in a meaningful way.

**Consider Solution:**

Shared case data support

**Related to Pattern:**

Case to case

B.3.5 Bypassing gateways

Data bypasses gateways because it is not needed until some later point in the execution
B.3 Internal:
B.3.6 Dangling inputs and outputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consider Issue:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bypassing synchronizing gateways</strong></td>
<td>Data</td>
<td>Internal</td>
<td>PAP-5.6</td>
</tr>
<tr>
<td>PLEASE SEE earlier description</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**B.3.6 Dangling inputs and outputs**

Inputs and outputs of an activity or gateway that remain unconnected in the model, very often as residues of connections that users decide to delete or redirect. (Many BPM tools/languages will catch this automatically, but not all)

**Consider Issue:**

**Dangling inputs can cause task to hang**

Dangling inputs are often the source of simulation errors or prevent the simulation from running all, because an activity or gateway waits for some input that it can never receive.

**Related to Solution:**

Use dangling inputs and outputs to reduce clutter.

**Dangling output means lost data**

Dangling data outputs show that a task or subprocess produced some data but this data is not used subsequently in the process.

**Related to Solution:**

Use dangling inputs and outputs to reduce clutter.

**Strongly redundant data**

A data element is strongly redundant if there is a writing activity after which in all possible continuations of the execution is completed. Strongly redundant data indicates in most situations a real flaw in the workflow.
B.4 Robustness:
B.4.1 Existence check

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related to Smell:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data is written to, but never used</td>
<td>Data</td>
<td>Usage</td>
<td>DAP-2</td>
</tr>
<tr>
<td>Weakly redundant data</td>
<td>Data</td>
<td>Usage</td>
<td>DAP-3</td>
</tr>
</tbody>
</table>

A data element is weakly redundant if there is some execution scenario in which it is written but never read afterwards, i.e. before it is destroyed or the total case's execution is completed. Weakly redundant data is not necessarily a flaw. For example, to improve performance some data may be computed in advance, even though it may not be needed later.

**Related to Smell:**
Data is written to, but never used  Data Usage  DAP-2

B.4 Robustness:

B.4.1 Existence check

Checking if a value or object currently exists.

**Consider Issue:**
Detecting undefined vs empty/null

Being able to differentiate between data elements that have an undefined value and those that are merely empty or null.

**Consider Solution:**
Deciding undefined vs empty/null

**Related to Pattern:**
Task precondition - data existence
Task postcondition - data existence
Task postcondition - data value
Task precondition - data value
B.4 Robustness:
B.4.1 Existence check

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A thread of control never arrives (lost tokens)</td>
<td>Control</td>
<td>Robustness</td>
<td>WCP-3</td>
</tr>
</tbody>
</table>

The use of the Synchronization pattern can potentially give rise to a deadlock in the situation where one of the incoming branches fails to deliver a thread of control to the join construct.

**Consider Solution:**

Structured process (no interrupts/exceptions) | Control | Usage | WCP-3
Cancel region | Control | Cancellation | WCP-29

**Related to Pattern:**

Synchronization (m-of-m and-join) | Control | Synchronizor | WCP-3
*(also known as: AND-join, rendezvous, synchronizer)*

Thread merge (and-join) | Control | Discriminator | WCP-41
Structured synchronizing merge (or-join) | Control | Synchronizor | WCP-7
*(also known as: Synchronizing join, synchronizer, OR-join)*

Generalized and-join (m-of-m and-join) | Control | Synchronizor | WCP-33
Local synchronizing merge (or-join) | Control | Synchronizor | WCP-37

**Missing tokens**

Individual abended threads/tokens would prevent a Join from resetting and failure to receive input on all of the incoming branches may result in the process instance (and possibly other process instances) stalling. Missing tokens could be a consequence of: - a design error - or that one of the tasks in the branch failing to complete successfully (e.g. as a consequence of it experiencing some form of exception) - or because the thread of control is passed outside of the branch. - or there are preceding splits or joins or they are un-structured in form (i.e. un-balanced splits and joins) (from WCP3)
B.4 Robustness:
B.4.1 Existence check

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel region</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-29</td>
</tr>
<tr>
<td>Cancelling partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Robustness</td>
<td>WCP-30</td>
</tr>
</tbody>
</table>

**Related to Pattern:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-30</td>
</tr>
<tr>
<td>Structured discriminator (1-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-9</td>
</tr>
</tbody>
</table>

*(also known as: 1-out-of-M join.)*

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static partial join for multiple instances (n:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-34</td>
</tr>
<tr>
<td>Multiple instances with a priori run-time knowledge (m:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-14</td>
</tr>
<tr>
<td>Multiple instances with a priori design-time knowledge (m:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-13</td>
</tr>
<tr>
<td>Multiple instances without a priori run-time knowledge</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-15</td>
</tr>
<tr>
<td>Dynamic partial join for multiple instances</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-36</td>
</tr>
<tr>
<td>Blocking partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-31</td>
</tr>
<tr>
<td>Blocking discriminator (1-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-28</td>
</tr>
<tr>
<td>Cancelling partial join for multiple instances (n:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-35</td>
</tr>
</tbody>
</table>

**Related to Smell:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the token design pattern</td>
<td>Data</td>
<td>State</td>
<td>CBP-4.2</td>
</tr>
</tbody>
</table>

**Bypassing synchronizing gateways**

Dataflow bypassing synchronization gateways of alternative branches can cause deadlocks because data needed by a future operation may not be created along some branch. Note: Bypassing forks and joins in parallel flows
B.4 Robustness:
B.4.1 Existence check

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>does not cause a deadlock, because all branches execute in parallel and thus, all business items always arrive through the connections i.e., all tasks receive their inputs as specified.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Consider Solution:**

Make input optional

**Related to Smell:**

Separate data and control flow channels

Bypassing gateways

**Related to Solution:**

Explicit data passing via data channels

**Discriminator missing tokens**

Individual abended threads/tokens would prevent a Discriminator from resetting and failure to receive input on all of the incoming branches may result in the process instance (and possibly other process instances) stalling.

**Consider Pattern:**

Cancelling discriminator (1-of-m and-join)

Cancelling partial join (n-of-m and-join)

**Related to Pattern:**

Structured discriminator (1-of-m and-join)

(Also known as: 1-out-of-M join.)

Structured partial join (n-of-m and-join)

Static partial join for multiple instances (n:m &-join)

Multiple instances with a priori run-time knowledge (m:m and-join)
B.5 Routing:

B.5.1 Data-based routing

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple instances with a priori design-time knowledge (m:m and-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-13</td>
</tr>
<tr>
<td>Multiple instances without a priori run-time knowledge</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-15</td>
</tr>
<tr>
<td>Dynamic partial join for multiple instances</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-36</td>
</tr>
<tr>
<td>Blocking discriminator (1-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-28</td>
</tr>
<tr>
<td>Blocking partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-31</td>
</tr>
<tr>
<td>Cancelling partial join for multiple instances (n:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-35</td>
</tr>
</tbody>
</table>

B.5 Routing:

B.5.1 Data-based routing

Data Routing WDP-40

When the evaluation of data-based expressions have the ability to alter the control-flow within a case.

Consider Issue:

No valid choice

Data Robustness WCP-4

If there are multiple options available, then at least one outgoing branch needs to be valid. If this is not the case, then there is the potential for the process to hang.

Consider Solution:

Default choice

Control Robustness WCP-4

Related to Pattern:

Exclusive choice (xor-split)

Control Split WCP-4
(Also known as: XOR-split, exclusive OR-split, conditional routing, switch, decision, case statement.)

Data-based routing

Data Routing WDP-40
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-choice (or-split)</td>
<td>Control</td>
<td>Split</td>
<td>WCP-6</td>
</tr>
<tr>
<td>(also known as: Conditional routing, selection, OR-split, multiple choice.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deferred choice (time-based xor-split)</td>
<td>Control</td>
<td>State</td>
<td>WCP-16</td>
</tr>
<tr>
<td>(also known as: External choice, implicit choice, deferred XOR-split.)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

More than one exclusive choices

If there are multiple valid options available what do you do if you can only choose one.

Consider Solution:

- Restricted choice
  - Control
  - Split
  - WCP-4

- Multi-choice
  - Control
  - Split
  - WCP-4

Related to Pattern:

- Exclusive choice (xor-split)
  - Control
  - Split
  - WCP-4

(Also known as: XOR-split, exclusive OR-split, conditional routing, switch, decision, case statement.)

- Data-based routing
  - Data
  - Routing
  - WDP-40

- Multi-choice (or-split)
  - Control
  - Split
  - WCP-6

(Also known as: Conditional routing, selection, OR-split, multiple choice.)

- Deferred choice (time-based xor-split)
  - Control
  - State
  - WCP-16

(Also known as: External choice, implicit choice, deferred XOR-split.)

B.6 State:

B.6.1 Using the token design pattern

A token is an exclusively held resource that is analogous to a physical object. Only one party can hold a token at any given time and if the token is destroyed, it cannot be used at all. This pattern is crucial to how many modeling languages work (e.g.
B.6 State:
B.6.1 Using the token design pattern

Name | Categ. | Section | Source
--- | --- | --- | ---
BPMN), however the pattern can be used within a Business Processes as well, for example: in formal meetings only the person who "has the floor" is supposed to speak.

**Consider Issue:**

**Missing tokens**

Individual abended threads/tokens would prevent a Join from resetting and failure to receive input on all of the incoming branches may result in the process instance (and possibly other process instances) stalling. Missing tokens could be a consequence of: - a design error - or that one of the tasks in the branch failing to complete successfully (e.g. as a consequence of it experiencing some form of exception) - or because the thread of control is passed outside of the branch. - or there are preceding splits or joins or they are un-structured in form (i.e. un-balanced splits and joins) (from WCP3)

**Consider Solution:**

Cancel region

Cancelling partial join (n-of-m and-join)

**Related to Pattern:**

Structured partial join (n-of-m and-join)

Structured discriminator (1-of-m and-join)

*(also known as: 1-out-of-M join.)*

Static partial join for multiple instances (n:m &-join)

Multiple instances with a priori run-time knowledge (m:m &-join)

Multiple instances with a priori design-time knowledge (m:m &-join)
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple instances without a priori run-time knowledge</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-15</td>
</tr>
<tr>
<td>Dynamic partial join for multiple instances</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-36</td>
</tr>
<tr>
<td>Blocking partial join (n-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-31</td>
</tr>
<tr>
<td>Blocking discriminator (1-of-m and-join)</td>
<td>Control</td>
<td>Discriminator</td>
<td>WCP-28</td>
</tr>
<tr>
<td>Cancelling partial join for multiple instances (n:m &amp;-join)</td>
<td>Control</td>
<td>Multiple_instance</td>
<td>WCP-35</td>
</tr>
</tbody>
</table>

A thread of control never arrives (lost tokens)

The use of the Synchronization pattern can potentially give rise to a deadlock in the situation where one of the incoming branches fails to deliver a thread of control to the join construct.

**Consider Solution:**

- Structured process (no interrupts/exceptions)                     | Control    | Usage         | WCP-3    |
- Cancel region                                                     | Control    | Cancellation  | WCP-29   |

**Related to Pattern:**

- Synchronization (m-of-m and-join)                                  | Control    | Synchronizer  | WCP-3    |
  *(also known as: AND-join, rendezvous, synchronizer)*
- Thread merge (and-join)                                            | Control    | Discriminator | WCP-41   |
- Structured synchronizing merge (or-join)                           | Control    | Synchronizer  | WCP-7    |
  *(also known as: Synchronizing join, synchronizer, OR-join)*
- Generalized and-join (m-of-m and-join)                             | Control    | Synchronizer  | WCP-33   |
- Local synchronizing merge (or-join)                                | Control    | Synchronizer  | WCP-37   |

**Detecting undefined vs empty/null**

Being able to differentiate between data elements that have an undefined
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>value and those that are merely empty or null.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Consider Solution:**
Deciding undefined vs empty/null

**Related to Pattern:**
Task precondition - data existence
Task postcondition - data existence
Task postcondition - data value
Task precondition - data value

**Related to Smell:**
Existence check

**Bypassing synchronizing gateways**
Dataflow bypassing synchronization gateways of alternative branches can cause deadlocks because data needed by a future operation may not be created along some branch. Note: Bypassing forks and joins in parallel flows does not cause a deadlock, because all branches execute in parallel and thus, all business items always arrive through the connections i.e., all tasks receive their inputs as specified.

**Consider Solution:**
Make input optional

**Related to Smell:**
Separate data and control flow channels
Bypassing gateways

**Related to Solution:**
Explicit data passing via data channels

**Discriminator missing tokens**
Individual abended threads/tokens would prevent a Discriminator from
resetting and failure to receive input on all of the incoming branches may result in the process instance (and possibly other process instances) stalling.

**Consider Pattern:**

Cancelling discriminator (1-of-m and-join)

Cancelling partial join (n-of-m and-join)

**Related to Pattern:**

Structured discriminator (1-of-m and-join)

(Also known as: 1-out-of-M join.)

Structured partial join (n-of-m and-join)

Static partial join for multiple instances (n:m and-join)

Multiple instances with a priori run-time knowledge (m:m and-join)

Multiple instances with a priori design-time knowledge (m:m and-join)

Multiple instances without a priori run-time knowledge

Dynamic partial join for multiple instances

Blocking discriminator (1-of-m and-join)

Blocking partial join (n-of-m and-join)

Cancelling partial join for multiple instances (n:m &-join)

**Leaving tokens behind**

If cancellation of a case is enabled, it is assumed that precisely one of the cancelling transitions will fire cancelling all necessary enabled tasks. To
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>achieve this, it is necessary that none of the cancelling transitions represent a state that is a superset of another possible state, otherwise tokens may be left behind after the cancellation. For example: During a mortgage application, the purchaser decides not to continue with a house purchase and withdraws the application. Someone needs to inform the property appraiser not to do an appraisal.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick kill</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-20-</td>
</tr>
<tr>
<td>By-pass</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-20-</td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel case</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-20</td>
</tr>
<tr>
<td><em>(also known as: Withdraw case.)</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel multiple instance activity</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-26</td>
</tr>
<tr>
<td>Cancel region</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-25</td>
</tr>
<tr>
<td>Impact to interdependent activities</td>
<td>Resource</td>
<td>Robustness</td>
<td>WRP-34</td>
</tr>
<tr>
<td><strong>Cancelling tokens from outside region</strong></td>
<td>Control</td>
<td>Usage</td>
<td>WCP-29</td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel region</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-25</td>
</tr>
<tr>
<td>Impact to interdependent activities</td>
<td>Resource</td>
<td>Robustness</td>
<td>WRP-34</td>
</tr>
<tr>
<td><strong>Unfinished business</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If the environment is not returned to the way it was found, this may cause problems for future activities and the cause of the problems may be hard to determine. For example: did the process reserve resources managed by other processes that need to informed they are no longer needed, etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cancel task</td>
<td>Control</td>
<td>Cancellation</td>
<td>WCP-19</td>
</tr>
<tr>
<td><em>(also known as: Withdraw task.)</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explicit termination</td>
<td>Control</td>
<td>Termination</td>
<td>WCP-43</td>
</tr>
<tr>
<td><strong>Not deleted on time</strong></td>
<td>Data</td>
<td>Performance</td>
<td>DAP-9</td>
</tr>
<tr>
<td>If large amounts of data are being processed, then unnecessary storage may be tied up if the data is not deleted immediately after it is no longer needed, even if it is meaningless without its original context, or it will be automatically cleaned up later (for example when the software program exits)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Related to Smell:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not cleaning up data</td>
<td>Data</td>
<td>Termination</td>
<td>DAP-7</td>
</tr>
<tr>
<td><strong>Never destroyed</strong></td>
<td>Data</td>
<td>Termination</td>
<td>DAP-7</td>
</tr>
<tr>
<td>If there is a scenario in which data is created but not destroyed afterwards, it may continue to exist and use up resources after the workflow case completes. If the data is sensitive this could become a security or privacy issue.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Related to Smell:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not cleaning up data</td>
<td>Data</td>
<td>Termination</td>
<td>DAP-7</td>
</tr>
<tr>
<td><strong>Token backlog</strong></td>
<td>Data</td>
<td>Robustness</td>
<td>tbd</td>
</tr>
<tr>
<td>Too many tokens (Threads of Control) waiting to be processed - for whatever reason. If there isn't a way to keep track of them all, some may be lost causing additional problems to tasks that needed them to complete. <em>(From Börger et al. 2008)</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Related to Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wait to be processed</td>
<td>Control</td>
<td>Trigger</td>
<td>WCP3</td>
</tr>
</tbody>
</table>
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for data</td>
<td>Control</td>
<td>Trigger</td>
<td>WDP-9</td>
</tr>
<tr>
<td>Data overload</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-11</td>
</tr>
</tbody>
</table>

**Consider Solution:**

- Allow block to create more data elements to handle overload
- Bind sub-workflow data to block task data

**Related to Pattern:**

- Sub-workflow decomposition to block task
- Block task to sub-workflow decomposition

**Related to Solution:**

- Merge changes strategy

**Transfer is not atomic.**

If control of something depends having possession of an object (for example a baton, or a crown) but the object is only exists digitally (e.g. a token), then it is possible for that object to exist in more than one place or cease to exist completely. For example, when transferring money electronically, the value must be deleted from one account and added to another. During this time, or if something went wrong, then the money could be in two accounts or no longer existing at all.

**Consider Solution:**

- Clean-up after transfer

**Related to Pattern:**

- The token design pattern
B.6 State:
B.6.1 Using the token design pattern

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object assumed to be protected</strong></td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-2.2</td>
</tr>
<tr>
<td><em>(also known as: Code assumed to be protected)</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assuming access by others is prevented when it actually isn't (like people who assume a bathroom door is always locked when the bathroom is in use)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Related to Smell:**

Using a lock when making changes to something shared

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wrong lock or no lock</strong></td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.2.3</td>
</tr>
<tr>
<td>If access to something is only controlled by an agreed upon &quot;protocol&quot; that has no way to absolutely control that access, then problems may occur when the protocol is not followed correctly or at all. For example people who are used to locked bathroom doors showing occupancy vs people who expect others to knock before opening the door.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Non-protected version is used instead of protected version**

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Control</td>
<td>Usage</td>
<td>CBP-4.1</td>
</tr>
<tr>
<td>If not all the ways to access something are protected then using the unprotected access (unnecessarily) may cause problems. For example always accessing a computer through an administrator (or root) account can cause a lot of problems.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Related to Pattern:**

Protection and subclassing

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-4.1</td>
</tr>
<tr>
<td><em>(also known as: Confinement and Subclassing)</em></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Double-checked locking**

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data</td>
<td>Creation</td>
<td>CBP-2.2.4</td>
</tr>
<tr>
<td>This comes from software, where a newly created object is available for others to use before it has been completely initialized. Similarly a report could be released for others to use while the author is still gathering information.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.7 Termination:
B.7.1 Multiple termination points

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonatonic operations assumed to be atomic</td>
<td>Control</td>
<td>Concurrency</td>
<td>CBP-2.2.1</td>
</tr>
</tbody>
</table>

Sometimes an activity that looks like it cannot be interrupted (self contained, i.e. atomic) can be because of the underlying activities which implement it. For example asking one person to sign a report not realizing they are really the contact person into an organization with its own approval process.

**Consider Pattern:**

- Critical section
- Interleaved routing
- Transfer by reference - with lock
- Two-stage access bug pattern

Sometimes a sequence of operations needs to be protected but instead it was wrongly assumed that protecting each operation separately is enough. For example transferring money electronically from one bank to another. It is not desirable if at any time that person appeared to have twice the amount of money or none.

**B.7 Termination:**

B.7.1 Multiple termination points

(Also known as: Stop Node in Parallel Execution Branches)

In some modeling languages there are stop nodes which shutdown the whole process as well as end nodes which just terminate an execution branch. The stop nodes may also be needed to release data from a sub-process to its parent process when it finishes. Therefore a model might contain stop/shutdown nodes on multiple branches.
B.7 Termination:
B.7.2 Not cleaning up data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consider Issue:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not all control paths may be ready to</td>
<td>Control</td>
<td>Termination</td>
<td>PAP-7.1</td>
</tr>
<tr>
<td>terminate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop nodes ending parallel branches always</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>terminate the whole process, even if tasks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>have not finished their execution, which</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>may leave shared data in an incomplete (i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e. corrupted) state.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Consider Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implicit termination</td>
<td>Control</td>
<td>Termination</td>
<td>WCP-11</td>
</tr>
<tr>
<td><strong>Consider Solution:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create single termination point.</td>
<td>Control</td>
<td>Termination</td>
<td>WCP-43</td>
</tr>
<tr>
<td>Use a single end or stop node,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Related to Pattern:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explicit termination</td>
<td>Control</td>
<td>Termination</td>
<td>WCP-43</td>
</tr>
</tbody>
</table>

B.7.2 Not cleaning up data

Data is not cleaned up after use, or ever

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consider Issue:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Never destroyed</td>
<td>Data</td>
<td>Termination</td>
<td>DAP-7</td>
</tr>
<tr>
<td>If there is a scenario in which data is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>created but not destroyed afterwards, it</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>may continue to exist and use up resources</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>after the workflow case completes. If the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data is sensitive this could become a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>security or privacy issue.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Not deleted on time</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If large amounts of data are being</td>
<td>Data</td>
<td>Performance</td>
<td>DAP-9</td>
</tr>
<tr>
<td>processed, then unnecessary storage may</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>be tied up if the data is not deleted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>immediately after it is no longer needed,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>even if it is meaningless with out its</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>original context, or it will be</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>automatically cleaned up later (for</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>example when the software program</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.7 Termination:
B.7.2 Not cleaning up data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>exits)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Unfinished business**

If the environment is not returned to the way it was found, this may cause problems for future activities and the cause of the problems may be hard to determine. For example: did the process reserve resources managed by other processes that need to informed they are no longer needed, etc.

**Related to Pattern:**

Cancel task

(\textit{also known as: Withdraw task.})

Explicit termination

**Leaving tokens behind**

If cancellation of a case is enabled, it is assumed that precisely one of the cancelling transitions will fire cancelling all necessary enabled tasks. To achieve this, it is necessary that none of the cancelling transitions represent a state that is a superset of another possible state, otherwise tokens may be left behind after the cancellation. For example: During a mortgage application, the purchaser decides not to continue with a house purchase and withdraws the application. Someone needs to inform the property appraiser not to do an appraisal.

**Consider Solution:**

Quick kill

By-pass

**Related to Pattern:**

Cancel case

(\textit{also known as: Withdraw case.})

Cancel multiple instance activity

Cancel region
B.8 Transfer:
B.8.1 Passing data to a different data type

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact to interdependent activities</td>
<td>Resource</td>
<td>Robustness</td>
<td>WRP-34</td>
</tr>
</tbody>
</table>

**Related to Smell:**

Using the token design pattern

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBP-4.2</td>
</tr>
</tbody>
</table>

Cancelling tokens from outside region

**Related to Pattern:**

Cancel region

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCP-29</td>
</tr>
</tbody>
</table>

Impact to interdependent activities

**B.8 Transfer:**

B.8.1 Passing data to a different data type

The Sender and Receiver use different data types when passing information

**Consider Issue:**

Different input data types

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDP-32</td>
</tr>
</tbody>
</table>

When applying a transformation function to a data element prior to it being passed to another process component, the transformation function has to have access to the same data elements as the receiving process component. It is also possible for translation errors to occur.

**Consider Pattern:**

Transformation - input

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDP-32</td>
</tr>
</tbody>
</table>

Transformation - output

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDP-33</td>
</tr>
</tbody>
</table>

**Related to Pattern:**

Transfer by value - incoming

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDP-27</td>
</tr>
</tbody>
</table>

Transfer - copy in/copy out

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDP-29</td>
</tr>
</tbody>
</table>

Transfer by value - outgoing

<table>
<thead>
<tr>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDP-28</td>
</tr>
</tbody>
</table>
B.9 Trigger:
B.9.1 Events as control flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different output data types</td>
<td>Data</td>
<td>Transfer</td>
<td>WDP-33</td>
</tr>
</tbody>
</table>

When applying a transformation function to a data element immediately prior to it being passed out of a process component, the transformation function has access to the same data elements as the process component that initiates it. It is also possible for translation errors to occur.

**Consider Pattern:**

- Transformation - output
- Transformation - input

**Related to Pattern:**

- Transfer by value - outgoing
- Transfer - copy in/copy out
- Transfer by value - incoming

117
B.10 Usability:
B.10.1 Multiple connections between activities

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data</td>
<td>Usability</td>
<td>PAP-5.3</td>
</tr>
</tbody>
</table>

Complex control and data flows easily lead to multiple connections in process models, which can result in cluttered models.

**Consider Issue:**

Complicated models are hard to maintain and can hide errors

**Consider Solution:**

Use dangling inputs and outputs to reduce clutter.

Use as few elements in the model as possible

Decompose a model with more than 50 elements

Minimize the routing paths per element

Reduce mixing data and control flow in a model.

Use one start and one end event

Model as structured as possible

Avoid or routing elements

Use verb-object activity labels

**Related to Smell:**

Cluttered data-flow models
B.11 Usage:

B.10.2 Cluttered data-flow models

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple connections between activities can result in errors</td>
<td>Data Usability</td>
<td>PAP-5.3</td>
<td></td>
</tr>
</tbody>
</table>

Multiple connections, for short multi-connections all start in the same activity or gateway and all end in one other activity or gateway. These connections lead to unnecessary redundancy if the multi-connections only involve control flow. If the multi-connections are associated with the same business item, they can easily lead to modeling errors. Data multi-connections usually point to a modeling problem where users either tried to pass the same item to several activities or intended to express that two different instances of the item are passed.

**Consider Solution:**

Reduce mixing data and control flow in a model.  
Minimize the routing paths per element

<table>
<thead>
<tr>
<th>Consider Issue:</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complicated models are hard to maintain and can hide errors</td>
<td>Data</td>
<td>Usability</td>
<td>PMG</td>
</tr>
</tbody>
</table>

PLEASE SEE earlier description

B.10.2 Cluttered data-flow models

**Consider Issue:**

Complicated models are hard to maintain and can hide errors

Data Usability PMG

B.11 Usage:

B.11.1 Data is written to, but never used

**Consider Issue:**

**Strongly redundant data**

A data element is strongly redundant if there is a writing activity after which in all possible continuations of the execution is completed. Strongly redundant data indicates in most situations a real flaw in the workflow.
B.11 Usage:
B.11.2 Data is over-written without being used first

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dangling output means lost data</td>
<td>Data</td>
<td>Internal</td>
<td>PAP-5.1</td>
</tr>
<tr>
<td>Dangling data outputs show that a task or subprocess produced some data but this data is not used subsequently in the process.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Related to Smell:**

<table>
<thead>
<tr>
<th>Related to Solution:</th>
<th>Categ.</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dangling inputs and outputs</td>
<td>Data</td>
<td>PAP-5.1</td>
</tr>
<tr>
<td>Use dangling inputs and outputs to reduce clutter.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Weakly redundant data**

<table>
<thead>
<tr>
<th>Weakly redundant data</th>
<th>Categ.</th>
<th>Usage</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A data element is weakly redundant if there is some execution scenario in which it is written but never read afterwards, i.e. before it is destroyed or the total case's execution is completed. Weakly redundant data is not necessarily a flaw. For example, to improve performance some data may be computed in advance, even though it may not be needed later.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B.11.2 Data is over-written without being used first

<table>
<thead>
<tr>
<th>Consider Issue:</th>
<th>Categ.</th>
<th>Usage</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly lost data</td>
<td>Data</td>
<td>Usage</td>
<td>DAP-4</td>
</tr>
<tr>
<td>A data element is strongly lost if there is a writing activity after which in all possible execution paths this element gets overwritten without having been read first. This result in lost resources if the data element is a reference to other data, or it may only be a performance problem.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weakly lost data</th>
<th>Categ.</th>
<th>Usage</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>A data element is weakly lost if there is an execution sequence in which it is overwritten without been read first. This may not be a flaw, for example capturing data in advance which may be updated at any time before it is actually used.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B.12 Visibility:
B.11.3 Data is read or destroyed before being initialized or written to.

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.11.3 Data is read or destroyed before being initialized or written to.</td>
<td>Data</td>
<td>Usage</td>
<td>DAP-1</td>
</tr>
</tbody>
</table>

**Consider Issue:**

**Missing data**

Data Usage DAP-1

The situation where some data element needs to be accessed, i.e. read or destroyed, but either it has never been created or it has been deleted without having been created again.

**Twice destroyed**

Data Termination DAP-8

A data element is twice destroyed if there is a scenario in which it is destroyed twice in a row without having been created in between.

B.12 Visibility:

B.12.1 Reusing names for data

Data Visibility WDP-1

Reusing names for data elements in different parts of the process model.

**Consider Issue:**

**Global name clash**

Data Visibility WDP-1

Name clash can occur when a task names a data element the same as in another part of the model which can be accessed from within that task. This may cause an immediate error or data may be unintentionally over written or hidden from view without warning.

**Consider Solution:**

Task data scope

Data Visibility WDP-1

**Related to Pattern:**

Task data

Data Visibility WDP-1
B.12 Visibility:
B.12.1 Reusing names for data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit case data access</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-5</td>
</tr>
<tr>
<td>Cascading name clash</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-2</td>
</tr>
</tbody>
</table>

If sub-processes can contain sub-processes, etc., then if data elements are accessible by sub-process, are they likewise accessible by their sub-processes, potentially increasing the chances of clashes.

**Consider Solution:**

Task data scope
Uniquely name concurrent non-independent data

**Related to Pattern:**

Block data
Folder data name clash

**Consider Solution:**

Access only one folder at a time

**Related to Pattern:**

Folder data
Scope data

Data elements in the block and subprocess not independent

if data elements are accessible from both a task and its sub-processes, then the potential exists for concurrency issues to arise as tasks executing in parallel update the same data element.

**Consider Solution:**

Uniquely name concurrent non-independent data
B.12 Visibility:
B.12.1 Reusing names for data

<table>
<thead>
<tr>
<th>Name</th>
<th>Categ.</th>
<th>Section</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related to Solution:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Explicit data passing via parameters</td>
<td>Data</td>
<td>Internal</td>
<td>WDP-10</td>
</tr>
<tr>
<td>Scope data name clash</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-3</td>
</tr>
<tr>
<td>Consider Solution:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task data scope</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-1</td>
</tr>
<tr>
<td>Related to Pattern:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scope data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-3</td>
</tr>
<tr>
<td>Folder data</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-6</td>
</tr>
<tr>
<td>Subprocesses can access everything</td>
<td>Data</td>
<td>Visibility</td>
<td>WDP-10</td>
</tr>
</tbody>
</table>

Information is accessible by everything as soon as it is available, so data is passed to other tasks simply by being made available, hence concurrent tasks may try to make changes to the same data at unexpected times.

Consider Pattern:
Block data

Consider to Smell:
Concurrency management

Related to Solution:
Implicit data passing