

1 **Mackenzie-Nickull Architectural Patterns Meta Model**

2

3 **Version 0.91**

4

June 5, 2005

5

Authors:

6

Matthew MacKenzie, Adobe Systems <mattm@adobe.com>

7

Duane Nickull, Adobe Systems <dnickull@adobe.com>

8

9

Abstract:

10

11

This is a meta model for documenting use cases as input for software architecture. The goal is to capture the business patterns then abstract out the patterns to be used as the basis for input requirements for software systems architecture.

12

13

14

15

Section 2 documents a subset usable for documenting use cases. It is designed to facilitate a quick capture of use cases from business stakeholders, then subsequent substance may be added by more technical stakeholders (architects, systems engineers etc).

16

17

18

19

20

Status:

21

22

This document is in the public domain and may be used with no further restrictions other than it remain intact. This document is updated periodically on no particular schedule. Send comments to the authors. There is no warranty etc. Use at your own risk.

23

24

25

Table of Contents

27	1.1.1	Patterns	2
28	1.2	Audience	3
29	1.3	The Mackenzie-Nickull Architectural Patterns Meta Model (UML)	3
30	1.3.1	A Reference Model for Architectural Patterns	3
31	1.4	The Pattern Notation	5
32	1.4.1	Pattern Presentation	6
33	1.4.2	Implementation	9
34	1.4.3	Business Problem (Story?) Resolved	9
35	1.4.4	Specializations	10
36	1.4.5	Known Uses	10
37	1.4.6	Consequences	10
38	1.4.7	References	10
39	1.4.8	Pattern Meta Model Summary	10
40	2	Use Case Patterns Subset Meta Model	11
41	2.1.1	Name	11
42	2.1.2	Description	11
43	2.1.3	Problem	11
44	2.1.4	Context	11
45	2.1.5	Derived Requirements:	11
46	2.1.6	Solution and Design Goals	12
47	3	References	13
48	3.1	Normative	13

49

1.1.1 Patterns

51 Patterns are recurring solutions to recurring problems. A pattern is composed of a problem, the
 52 context in which the problem occurs and the solution to resolve this problem. The focus of this
 53 document is to illustrate a model to capture the structural organization of a system, relate that to
 54 its' requirements and highlight the key relationships amongst entities within the system. The
 55 patterns meta model illustrated in this document is meant for both lay-people as well as seasoned
 56 experts.

57

58 This specification is not dependent on any specific standards, although it advocates using UML
 59 for consistent representations of patterns. Other modeling syntaxes may be used. All patterns
 60 are agnostic to actual implementation detail with regard to specific technologies and no
 61 presumptions are made with regards to a bias towards any specific protocols or standards other
 62 than as noted or may be written by pattern authors.

63

64 The concept of patterns evolved from work by Christopher Alexander, a brilliant architect born in
 65 Vienna. He is the primary author of a book called "*A Pattern Language*"¹ which had a great
 66 influence on object oriented programming. The basic concept of the book was a realization that
 67 patterns are the same when architecting a city, a block, a house and a room. Each of these
 68 entities employs similar patterns.

69

70 In 1996 the book Pattern Oriented Software Architecture[[ISBN 0471958697 \(alternate, search\)](#)]
71 by Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal
72 defined an architecture pattern as:

73

74 "expressing a fundamental structural organization schema for software systems.

75 It provides a set of predefined subsystems, specifies their responsibilities, and

76 includes rules and guidelines for organizing the relationships between them."

77

78 The work continued to evolve by [Erich Gamma](#), [Richard Helm](#), [Ralph Johnson](#), and [John](#)
79 [Vlissides](#) – most famously referred to as the "Gang Of Four". They published a book called
80 "*Design Patterns: Elements of Reusable Object-Oriented Software*" that leveraged Christopher
81 Alexanders thinking and brought it to new heights in 1999. There are numerous references to
82 many patterns on the internet, however the Gang of Fours work is the most notorious.²

83

84 **1.2 Audience**

85

86 The audience members for this document may fulfill multiple roles in the development of software
87 solutions architecture. Some of the specific roles include, but are not limited to:

88

89 the business or enterprise analyst

90 those responsible for the operations of an enterprise

91 the systems analyst

92 the developer

93

94 The document provides a framework for the decision-maker to view the business from a different
95 perspective from the engineer.

96

97 This document is intended to facilitate the requirements gathering by the business analyst. The
98 meta model should guide the systems analyst in defining the logical boundaries of a solution to a
99 problem.

100

101 **1.3 The Mackenzie-Nickull Architectural Patterns Meta Model** 102 **(UML)**

103

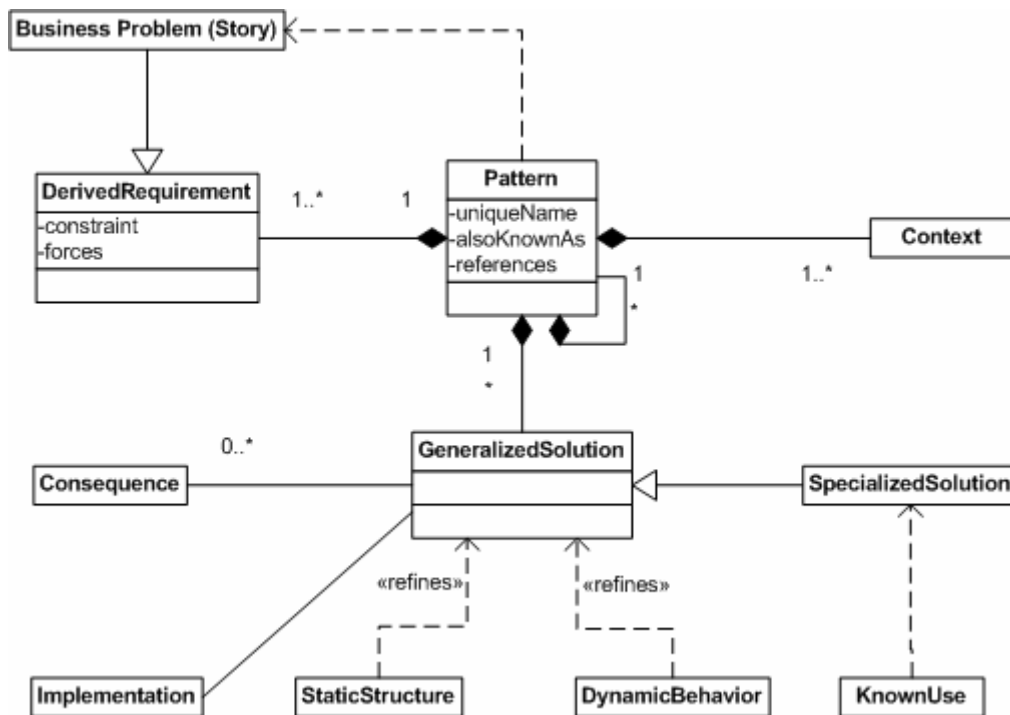
104 **1.3.1 A Reference Model for Architectural Patterns**

105

106 This meta model constrains instances of patterns. Specific patterns may extend the base model
107 in order to facilitate functionality required within a certain context. This meta model is a
108 specialization of the basic architectural patterns (Problem-Context-Solution) used by many
109 software developers in order to facilitate conveyance of concepts to the target audience.

110

111



112
113

114 Figure 1.3.1 –Pattern Meta Model (UML)

115

116 A pattern may be decomposed into three main components – the *Context*, the *Business Problem*
117 (expressed as *Derived Requirements*) and the *Generalized Solution*. All three parts of a pattern
118 are intrinsically inter-related. Patterns themselves may be composed of other patterns.

119

120 Context

121

122 The context is the set of circumstances or situation(s) in which the business problem. A
123 context may be highly generalized in order to ensure maximum applicability of the pattern or it
124 may be highly specialized. A highly specialized context may even be a specific instance's
125 context.

126

127 The context should not be considered an exhaustive set of contexts for which a problem may
128 occur. It is highly unlikely that a pattern developer could or would bother to envision ever single
129 conceivable context in which a specific problems occurs and document them. A more pragmatic
130 approach is to list all known situations where a problem addressed by the pattern occurs.

131

132 The primary goal of noting the context is to document as many external forces that affect the
133 pattern as possible.

134

135 Business Problem

136

137 The problem is a documentation of the problem(s) that arise repeatedly within the context(s) and
138 is augmented with a series of *forces*. A generalized description of the problem (detailed
139 according to requirements) must be present. A set of secondary artifacts may also be
140 documented – the requirements, constraints and desirable properties for a solution.

141

142 The use of different viewpoints to can be employed in order to facilitate greater comprehension of
143 a specific problem

144

145 Generalized Solution

146

147 The solution specifically resolves the recurring business problem and/or how to balance the
148 constraints and forces of the problem. The static structure of the generalized solution contains
149 concrete classes (objects) and relationships and is expressed using Unified Modelling Language
150 (UML) class view diagrams.

151

152 The dynamic behavior of the generalized solution documents how classes (objects or
153 components) collaborate between each other and captures the dependencies and relationships
154 between them. UML sequence diagrams are used to depict the dynamic behavior of the solution.

155

156 1.4 The Pattern Notation

157

158 This section describes the Pattern meta model in greater detail.

159

160 Individual patterns may be generally classified in the following manner:

161

162 **Architectural Patterns** – a fundamental structural organizational schema for software systems.
163 It provides a set of predefined subsystems specifies their responsibilities and includes rules and
164 guidelines for organizing the relationships between them.³

165

166 **Design Patterns** - provide a scheme for refining the subsystems or components of a software
167 system, of the relationships between them. It describes a commonly recurring structure of
168 communicating components that solves a general design problem within a [specific] context.¹

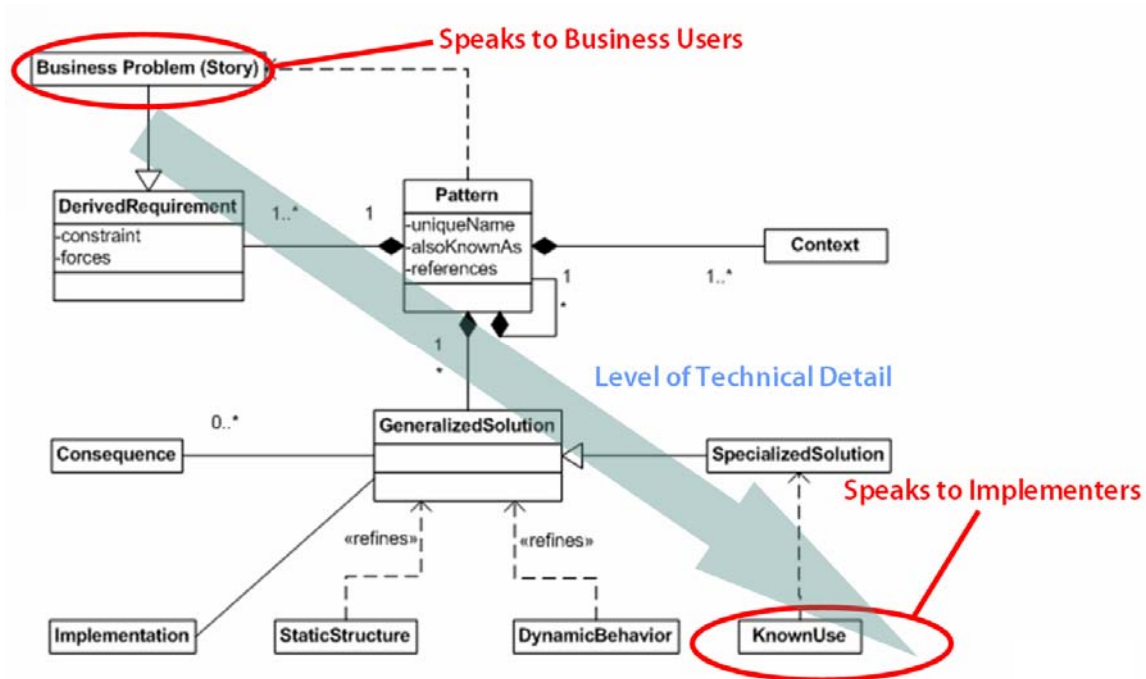
169

170 **Idioms** – Idioms are the lowest level patterns and may be specific to a programming language.
171 An idiom guides implementation aspects of components and the relationships between them
172 using features specific to a given language or environment.

173

174 The pattern notation speaks to both business analysts as well as technical audience member as
175 shown in the figure below.

176



177

178 1.4.1 Pattern Presentation

179

180 Each pattern instance SHOULD be composed of the following text and graphic components,
 181 according to the UML rules of cardinality.

182

183 1.4.1.1 Name

184

185 The Pattern name is a unique name (within the domain) accompanied by a short descriptive
 186 summary of the pattern.

187

188 1.4.1.2 Also known As (optional)

189

190 Alternative names for the pattern if any are known. See also "see also" later in this section.

191

192 1.4.1.3 Business Problem (Story)

193

194 The Business Problem is a specific, illustrative example of the problem to document the
 195 requirement for the pattern. This is similar to the architectural concept of "Story". There can be
 196 more than one story per pattern. The story also helps speak to the business user to identify that
 197 a specific pattern may be relevant to their situation.

198

199 **1.4.1.4 Context**

200 The context is the situation or set of unique situations in which the pattern may apply. The
201 context captures a specific context or a set of contexts in which the problem occurs. The higher
202 the degree of generality, the higher the likelihood the pattern may be reusable and vice versa.

203

204 **1.4.1.5 Derived Requirements**

205 The derived requirements are a summary of the Business Problem (Story). While the Business
206 Problem is often specific, the derived requirements are a generalized extrapolation of the general
207 concepts. A description of the problem including a list of any *constraints* associated with this
208 problem. Unlike the story component, the Derived Requirements specifically describe the
209 problem in detail.

210

211 An example would be where the Business problem is that an airline company must send an
212 electronic passenger manifest to its' head office and have reception of the message
213 acknowledged by their head office by having them send it back. The derived requirement could
214 be described as *"reliable electronic messaging"*.

215

216 **1.4.1.6 Generalized Solution**

217 The generalized solution for the problem as it occurs within the context. The solution can be
218 further subdivided into two different components – the overall structure of the solution as a static
219 diagram and its' dynamic behavior.

220

221 **1.4.1.7 Static Structure**

222

223 This section is a generalized description of the static components, their organization and the
224 relationships amongst them.

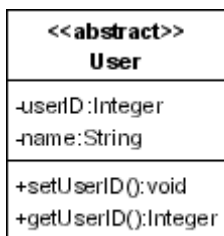
225

226 An example of a class view diagram is depicted in Figure 1.3.1 – the UML class view diagram
227 used to express a view of the Patterns Metamodel.

228

229 UML Class view diagrams show the classes, complete with their attributes and operations, and
230 are also capable of expressing the complexities of the relationships between classes. While the
231 lines are blurred between data objects and executable application logic, the context of each class
232 view diagram is important to capture.

233

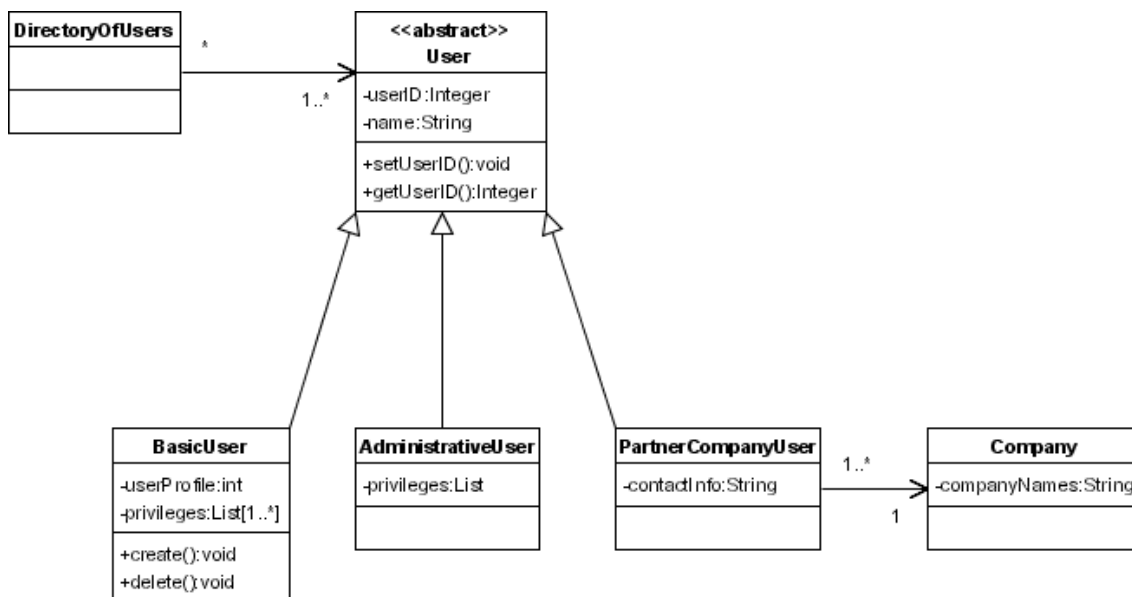


234

235 Figure 1.3.1.7 – single class UML class view diagram

236

237 Figure 1.3.1.7 (above) shows a simple UML class view diagram. The stereotype is a
 238 generalization of the type of the class, in this case an <<abstract>> class. The class has two
 239 attributes in the second section, a userID and a name. Each attribute also has an optional
 240 datatype. The third portion of the class is the operations of the class, in this case two public
 241 operations exist to set or get a userID.
 242



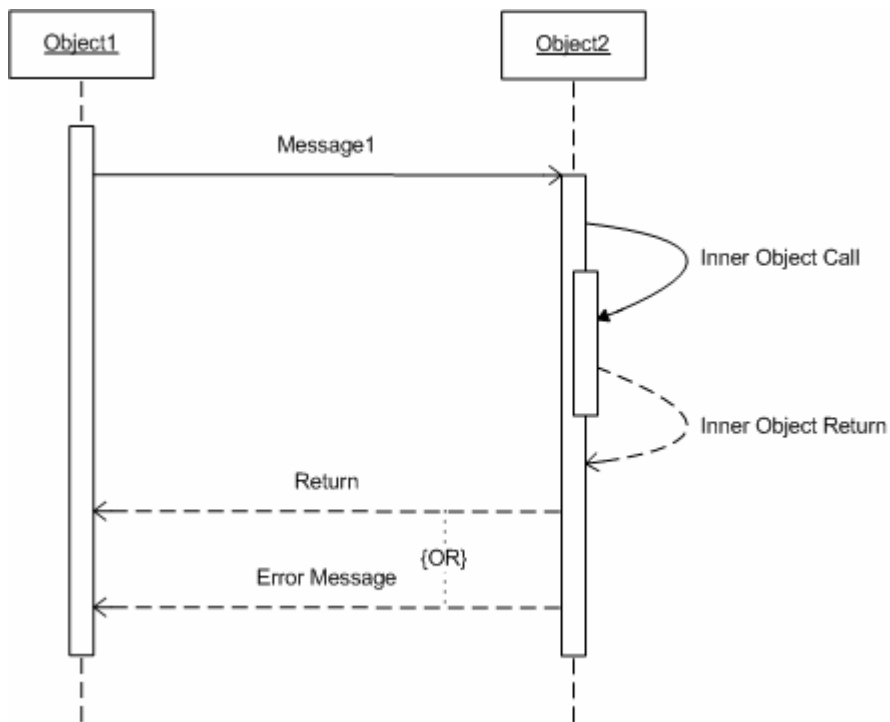
243
 244 Figure 1.3.1.8 – UML Class diagram example

245
 246 In the above figure, there are several classes on one class view diagram. The lines that connect
 247 the classes declare the relationships between them. Numbers indicate cardinality. For example,
 248 a user may belong to any number of directories between zero and infinity, as declared by the
 249 asterisk (*). A DirectoryOfUsers has one to infinity users as declared by the "1..*" (without at least
 250 one user, it may be debatable if it is a directory). The arrow shows that the DirectoryOfUsers is
 251 dependent upon Users. BasicUser, AdministrativeUser and PartnerCompanyUser all inherit from
 252 the abstract user class and are deemed specializations of user. The base abstract user class is
 253 considered a "generalization" of the other specific types of users.
 254

255 1.4.1.8 Dynamic Behavior

256 The dynamic behavior aspect of any pattern describes the runtime interactions, sequences and
 257 behavior of the pattern's solution. The use of the UML Sequence diagram syntax is used to
 258 depict these views. UML Sequence diagrams are a part of the Object Meta Group's Unified
 259 Modelling Language (UML) version 2.0 specification and considered a standard notation for
 260 specifying dynamic behavior among concurrently-operating objects and processes of hardware
 261 components.
 262

263 Within this architecture specification, we use UML Sequence diagrams to illustrate the
 264 relationships between components of the pattern during runtime.
 265



266
267 Figure 1.3.1.9 – UML Sequence Diagram example

268
269 The UML sequence diagram in the above figure shows two objects. The reader should interpret
270 the sequence diagrams starting from the top left hand corner. Object1 makes a call to Object2,
271 Object2 does some internal process then either sends a return or error back to Object1.

272
273 For more on UML Sequence or Class view diagrams, please visit <http://www.uml.org>.
274

275 1.4.2 Implementation

276 Each implementation section contains notes for implementing the pattern. Within this
277 specification, care is taken not to constrain this to any specific platform or programming language
278 or specific standard or protocol. The implementation section of each pattern is therefore
279 somewhat more vague than some patterns (specifically Idioms) viewers may be used to.

280 In general, the implementation section is considered a non-normative suggestion, not an
281 immutable rule or requirement. In terms of RFC 2119 interpretation, the implementation is of
282 RECOMMENDED or MAY status.

283
284 An example may be to note that a reliable messaging server should employ the use of a second
285 machine to detect its' heartbeat and take action should it fail to operate. Another example of
286 implementation detail may be that when a person registers for the user directory, another human
287 checks their credentials to ascertain their identity.

288

289 1.4.3 Business Problem (Story?) Resolved

290 This section of the pattern contains reconciliation between the solution and the business problem
291 (story) of the pattern. This section may contain additional details not yet covered by the solution
292 and implementation sections and their subsections.

293 **1.4.4 Specializations**

294 Specializations are specific or customized instances of the generalized solution. An example
295 could be to outline two specializations to building user interfaces. Such may be implemented as
296 a non-web based GUI (such as using the Microsoft Foundation Classes (MFC) in the
297 implementation section. The other specialization could alert readers to the fact that a web
298 interface could also be build to achieve the same end using Java Server Pages or a similar
299 technology.

300

301 **1.4.5 Known Uses**

302 The known uses are references to examples of the specializations of the pattern in use. For
303 example – the pattern could be the condition of an artifact that is used to capture the details of
304 how to bind to a specific web service. There could be three known uses – a Universal
305 Description Discovery Interface (UDDI) registry service binding method, a Web Services
306 Description Language (WSDL) instance or an ebXML Collaboration Profile Protocol (CPP)
307 instance. The exact difference of each known use may vary and may be discussed lightly in this
308 section to guide implementers.

309 **1.4.6 Consequences**

310 The benefits the pattern provides and any potential liabilities or caveats. This is an analysis of
311 the consequences imposed by the Generalized Solution. Consequences of specialized solutions
312 may also be discussed herein.

313 **1.4.7 References**

314 A set of reference to other patterns or information relevant to the problem, context and solution.

315

316 **1.4.8 Pattern Meta Model Summary**

317 The metamodel for patterns has been adopted from a variety of industry definitions of patterns.
318 Its' goal is to facilitate the capture and sharing of concepts and relationships between
319 components.

320

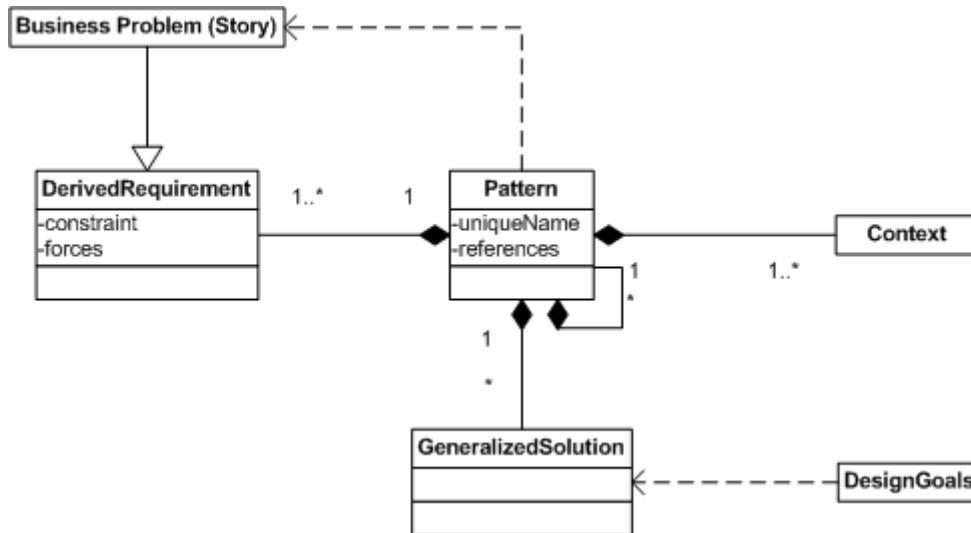
2 Use Case Patterns Subset Meta Model

321

322

323 There are many instances where patterns may need to be documented in simplistic form first,
324 then detail subsequently filled in by more skilled architects. It is RECOMMENDED that the
325 following subset of the patterns be used for capturing higher level patterns and use cases.

326



327

2.1.1 Name

329 The name of the use case pattern

2.1.2 Description

331

332 The brief description of the use case. Limit this to 2-4 sentences.

2.1.3 Problem

334

335 The problem described in business terms. Elaborate all aspects of the problem that may place
336 constraints upon a solution or that are relevant to derived requirements.

2.1.4 Context

338

339 The context in which this pattern occurs.

2.1.5 Derived Requirements:

341

342 This section may be left blank by the use case author and left for architects to fill out. The
343 concept is to abstract out requirements that are not tied to the specific use case.

344 **2.1.6 Solution and Design Goals**

345

346 This section may be left blank by the use case author and left for architects to fill out. This may
347 briefly illustrate how a solution may be derived. Architects may use this section to make notes on
348 how this use case affects or introduces design goals for the architecture.

349

350

351

352

3 References

353

3.1 Normative

354

[RFC2119]

S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
<http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

355

356

357

358

¹ Alexander, C., S. Ishikawa, & M. Silverstein, *A Pattern Language*, Oxford University Press, 1977.

² <http://c2.com/cgi/wiki?DesignPatternsBook>

³ "Pattern-Oriented Software Architecture" Buschmann, Meunier, Rohnert, Sommerlad and Stal – ISBN 0 471 95869 7