Pattern and Component Markup Language (PCML)

Draft 3

LICENSE INFORMATION

This document and its contents are furnished "as is" for informational purposes only, and are subject to change without notice. ObjectVenture Inc. does not represent or warrant that any product or business plans expressed or implied will be fulfilled in any way. Any actions taken by the user of this document in response to the document or its contents will be solely at the risk of the user.

OBJECTVENTURE MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, WITH RESPECT TO THIS DOCUMENT OR ITS CONTENTS, AND HEREBY EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR USE OR NON-INFRINGEMENT. IN NO EVENT SHALL OBJECTVENTURE BE HELD LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING FROM THE USE OF ANY PORTION OF THE INFORMATION.

TRADEMARKS

If you have any comments concerning this document or software, please forward them to:

ObjectVenture Inc.
89 Main Street
Third Floor
Milford, MA 01757
Internet address: info@objectventure.com

Table of Contents

Chapter 1

Introduction

The current state of software reuse and how this specification furthers reuse

Contents

Current state of reuse

Companies around the world are engaged in object-oriented and component-based development but are failing in their quest to reuse the components that they and others have created. The problem is that developers can create components themselves, but when they attempt to pass their work onto others, there is no standard mechanism for having the component describe itself, what it is designed to do, how it collaborates with other components and how its functionality may be extended. Developers who would benefit by leveraging an existing component find themselves tracking down the original producer to find out its inner workings. And after trudging through such a long exercise, often the component conflicts in some way with their needs.

A growing trend in software development is the emergence of patterns at multiple levels of abstraction and in multiple domains of knowledge. Patterns provide a way for system architects to convey best practices and design strategies to other architects and software developers that must build flexible and scalable software systems. In short, they further the idea of software design reuse.

Patterns are currently documented and exchanged using one of several simple plain text templates, diagrams and example code. However, this method is less than ideal. For patterns to be truly useful as a medium of reuse, approachable and available in the general software development community, the following issues must be addressed:

- Even if one standard template existed, the leap from a textual description to a concrete design is enough to discourage many from effectively leveraging patterns.
- The templates that do exist are mostly textual and do not lend themselves to tool automation.
- Most templates offer no consistent mechanism for realizing multiple strategies of the same pattern.
- Although the concept of a pattern catalog exists, there is no standard way of describing one.
- There is currently no standard way to describe a component's role in patterns.

Patterns and components share the following needs, which are not yet fully addressed in a standard way:

- Ownership, which includes a clear statement of intellectual property rights and licensing guidelines.
- Versioning, which is addressed by some component standards, but not by all and not in a standard way.
- Reference or inclusions of external artifacts that further describe the pattern or component.
- A mechanism for cataloging and packaging a number of patterns or components.

How this specification furthers reuse

This specification allows patterns at any level of abstraction to be expressed in a tangible, standard format. By leveraging the openness and flexibility of XML, this technology enables architects and developers to easily and effectively describe, package, exchange and extend their own patterns as well as those created by others. At the same time, tool and repository providers are empowered to automate much of this process.

This specification is aimed squarely at providing robust and intelligent tools for pattern-driven development of software frameworks and applications. It provides an explicit mechanism for describing how components participate in patterns, thereby showing the author's intent and easing the road to maintenance and reuse. Components become self-describing and application assembly much simpler.

Chapter 2

Overview

Provides a high-level description of the pattern and component metamodel

Contents

Pattern Overview

Patterns have a number of characteristics, some of which the pattern metamodel outlined in this specification must explicitly support:

- Patterns capture proven, reusable designs and implementations that aid in the prevention of unnecessary reinvention.
- Patterns often represent best practices that have been mined from the collective experience of the software development community.
- Patterns describe component collaborations that provide solutions to problems in a given context.
- Patterns exist at multiple levels of abstraction.
- Patterns address both functional and non-functional requirements.
- Patterns are arguably the most useful to system architects and developers when used in combination to solve complex, recurring problems.

While preserving the widely used context-problem-solution form of patterns that is often rooted in a textual description, the authors of this specification recognize the need for a more robust description of patterns to meet the characteristics enumerated above and in the previous chapter. A simplified graphic of the resulting metamodel is provided in Figure 1.

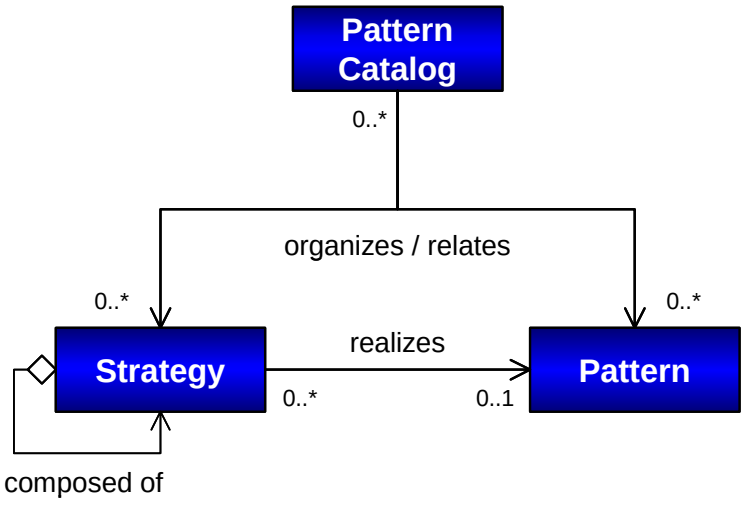

Figure 1: Overview of Pattern Metamodel

Each one of these elements is summarized below. A detailed explanation of each is provided in later sections.

Catalog

A side effect of documenting patterns is the outgrowth of a common language used to communicate their use. So instead of routinely getting mired down in details when trying to express a solution to a problem, just mentioning the names of certain patterns immediately conveys a deep description of a solution. This common language serves as a catalyst for more productive design discussions and knowledge transfer among architects and developers.

This notion of a common language is composed of domain-specific subsets commonly referred to as pattern catalogs. We adopt that term here as part of the pattern metamodel and assign to it the following functions:

- Catalogs are a grouping mechanism for a number of related patterns and their strategies. Although patterns are usually grouped within a domain, a catalog does not restrict groupings that span multiple domains.
- Catalogs provide a simple means of pattern classification.
- Catalogs facilitate the packaging and reuse of patterns and their strategies.
- A catalog may be composed of other catalogs.

Pattern

A pattern is a somewhat generic description of a solution provided to address one or a common set of design problems in a certain context. This specification recognizes a pattern as just that with no direct implementation details. In this sense, it serves as a class of solutions. The details of a particular solution are captured in a strategy, which is discussed later.

A pattern, then, is assigned the following functions:

- A pattern defines a context, a problem and a general solution. A solution here is an abstract description that is not tied to any particular implementation.
- A pattern defines participants and describes how they interact to provide a solution.
- A pattern may reference other patterns or external artifacts.

- Some patterns are not amenable to solutions that may be implemented. Therefore, a pattern is not required to have any strategies.

Strategy

Since patterns describe general solutions to problems, there is almost always more than one way to realize each one of them in software systems. Pattern authors usually include code samples and maybe even a complete example of at least one solution to aid in the use of a pattern.

We take a slightly different approach here by removing implementation details from the pattern itself and codifying them in a strategy. Each pattern may have multiple strategies, each of which defines one implementation of a pattern solution. A pattern isn't directly aware of its strategies because we do not wish to limit the number of strategies available and the association of them to a pattern at just creation time. It is conceivable that people other than the original author may later discover new strategies for applying a pattern.

A strategy, then, is assigned the following functions:

- A strategy may define one of many possible implementations of a pattern solution.
- A strategy defines one or more roles that may be mapped to concrete components and their elements.
- A strategy provides a mechanism for constraining which components and elements may fill each role.
- Patterns are often composed of other patterns. A strategy addresses this "pattern nesting" by being composed of other strategies. This scalability allows the description of large component collaborations or frameworks.
- A strategy is not required to be associated with a pattern. It may instead serve as a building block for other strategies or as an idiom.

Component Overview

The representation of a component and a mechanism for organizing a set of them is necessary for this specification to address the needs listed in the previous chapter. A simplified graphic of the resulting metamodel is provided in Figure 2.

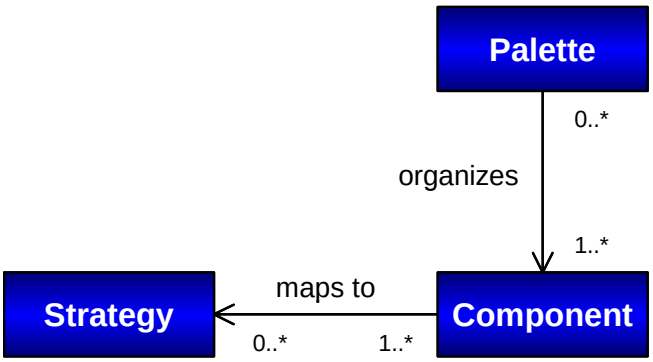

Figure 2: Overview of Component Metamodel

Each one of these elements is summarized below. A detailed explanation of each is provided in later sections.

Palette

The notion of a palette is widely used for organizing user interface components and, to a much lesser extent, business components. We adopt the term here and use it to provide a simple means of classification and organization. A palette is assigned the following functions:

- Palettes are a grouping mechanism for a number of related components.
- Palettes provide a simple means of component classification.
- Palettes facilitate the packaging and reuse of components and frameworks.
- A palette may be composed of other palettes.

Component

Pattern strategies describe a collaboration of components that provide a solution to a problem or that address a certain need (as described by a pattern). This specification does not attempt to describe components themselves, because that has already been done with some degree of success. It does describe how patterns and components interact.

A component, then, is assigned the following functions:

- A component may fill one or more pattern strategy roles in isolation or as part of any number of component collaborations.

- A component can describe who its author is, what version it is, what it does, and how it may be used.

Roles

This specification defines five distinct roles in the development of software applications using patterns and components. Each of these roles adheres to a contract that ensures its product is compatible with the others.

The packaging requirements for each role are defined in *Chapter 11: Packaging Requirements*.

Pattern Provider

The Pattern Provider is the producer of patterns. This individual party is responsible for codifying a general solution to one or more related problems in a well-defined context. Documenting patterns is a difficult process that is typically the place of an experienced architect of software systems who is able to leverage that experience to identify and effectively communicate candidate patterns.

A pattern solution may have one or more different implementations, each of which the Pattern Provider describes with a strategy. Strategies are commonly discovered over time as a pattern is used in different situations and by different people. In this case, the provider may choose to extend existing patterns created by another party by adding one or more additional strategies.

The Pattern Provider may hierarchically organize or classify a number of related patterns using catalogs. Descriptions may also be provided for common associations between patterns that aid in their use together.

The Pattern Provider's output generally consists of catalog JARs, which include catalog descriptor files, pattern descriptor files and a strategy descriptor file for each strategy the provider wishes to include for a pattern (optional). These descriptors are required to have '.catalog,' '.pattern' and '.strategy' extensions respectively.

Component Provider

The Component Provider is the producer of components. This individual party is responsible for creating components and packaging them according to any guidelines that exist for the chosen implementation technology. For example, the

construction and packaging of an Enterprise JavaBean™ (EJB™) should be done in compliance with the EJB specification.

In addition to the component itself, the Component Provider's output is a component descriptor that must be packaged with the component. The extension of this descriptor file must be '.component.' One descriptor for each component interface that is implemented by the component should also be included. If a particular component standard does not support packaging via JAR or ZIP archives, then a palette may be used for packaging purposes.

The Component Provider may hierarchically organize or classify a number of related components using palettes. Each palette is simply a logical container of components, which itself may be nested in another palette.

Tool Provider

The Tool Provider is the producer of tools that leverage this specification to automate the creation, exchange and use of patterns, components and frameworks.

Marketplace Provider

The Marketplace Provider is the producer of public and private marketplaces or repositories for the express purpose of mining, organizing, and exchanging patterns, components and frameworks.

Application Assembler

The Application Assembler is the producer of applications created using prepackaged patterns and components.

Chapter 3

General Elements

Provides a detailed specification of general elements used throughout the metamodel

Contents

Enumerated Types

The following enumerated types are commonly used throughout this specification, so we define them here to avoid duplication.

Boolean

A *Boolean* type represents one of two truth-values, True or False. Any attribute of this type may have any one of the following values listed in Table 1.

Table 1: Boolean Enumerated Type Values

Value	Description
true	The attribute has a truth-value of True.
false	The attribute has a truth-value of False.
yes	The attribute has a truth-value of True.
no	The attribute has a truth-value of False.

Access

An *Access* type represents the visibility of an element to the outside world. Any attribute of this type may have any one of the following values listed in Table 2.

Table 2: Access Enumerated Type Values

Value	Description
public	The element the attribute represents is visible to any other element that may reach it.
private	The element the attribute represents is only visible to that element's internals.
protected	The element the attribute represents is visible to that element's internals and to other elements it has a relationship with (including sub-elements).

Aggregation

An *Aggregation* type represents the nature of a relationship between two elements. Any attribute of this type may have any one of the following values listed in Table 3.

Table 3: Aggregation Enumerated Type Values

Value	Description
composition	The source element is composed of and owns the target element in the relationship and is responsible for its lifecycle.
aggregation	The source element is composed of, but does not own, the target element in the relationship.
none	This value must be used for both ends of a peer-level relationship, where neither element is composed of the other. It must also be used for the target of a composition or aggregation relationship.

Mutability

A *Mutability* type represents the changeability of an element. Any attribute of this type may have any one of the following values listed in Table 3.

Table 4: Mutability Enumerated Type Values

Value	Description
read	The element the attribute represents may be queried but not changed.
read-write	The element the attribute represents may be queried and changed.

URL

A *URL* element represents a link with a friendly display name. It may be used to represent an email address, a web page, etc. The attributes of a URL are defined in Table 5.

Table 5: Email Attributes

Value	Type	Required	Description
display-name	String	No	A display name or friendly label for the URL
address	String	Yes	The actual URL

Author

An *author* represents the creator of a pattern, component, artifact, etc. and is used to provide ownership. An author is defined by the attributes and associations enumerated in Table 6 and Table 7 respectively.

Table 6: Author Attributes

Value	Type	Required	Description
name	String	Yes	Name of the author
organization	String	No	Organization the author represents. If the name is an actual organization, then this attribute may be omitted.
description	String	Yes	Description of the author

Table 7: Author Associations

Element	Cardinality	Required	Description
URL	0..*	No	A URL where information pertaining to the author, his organization, or his works may be obtained. This may also represent an email address.

A graphic representation of an author's association with other elements is provided in Figure 3.

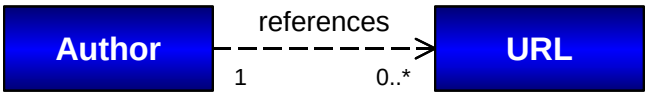

Figure 3: Author Associations

Version

A *version* represents the state of development an element is at. Its primary purpose is to distinguish multiple revisions of the same element.

A version is defined by the attributes and associations enumerated in Table 8 and Table 9 respectively.

Table 8: Version Attributes

Value	Type	Required	Description
revision	String	Yes	Version number
date	String	Yes	Date/time of the revision
description	String	No	Description of the revision
copyright	String	No	Copyright notice for the revision.
release-notes	String	No	Notes that describe important aspects of the revision
license	String	No	Licensing information for the revision.

Table 9: Version Associations

Element	Cardinality	Required	Description
Artifact	0..*	No	External documents or other resources that further describe the revision

A graphic representation of a version's association with other elements is provided in Figure 4.

Artifact

An *artifact* represents an external file that may not be appropriately supplied in the XML form defined in this specification. It helps to further describe what a particular element represents or instruct in its use. Examples of an artifact include a UML diagram (binary or XMI), a graphical image, documentation, etc.

An artifact is defined by the attributes and associations enumerated in Table 10 and Table 11 respectively.

Table 10: Artifact Attributes

Value	Type	Required	Description
name	String	Yes	Name of the artifact
type	String	No	File type of the artifact, which should be represented by a common file extension (i.e. html, doc, mdl)
description	String	No	Description of the artifact

Table 11: Version Associations

Element	Cardinality	Required	Description
Author	0..*	No	An author of the artifact
URL	1..*	Yes	Location of the artifact in the form of a URL, which may be either relative or absolute
Version	1	No	Version information for the artifact

A graphic representation of a version's association with other elements is provided in Figure 4.

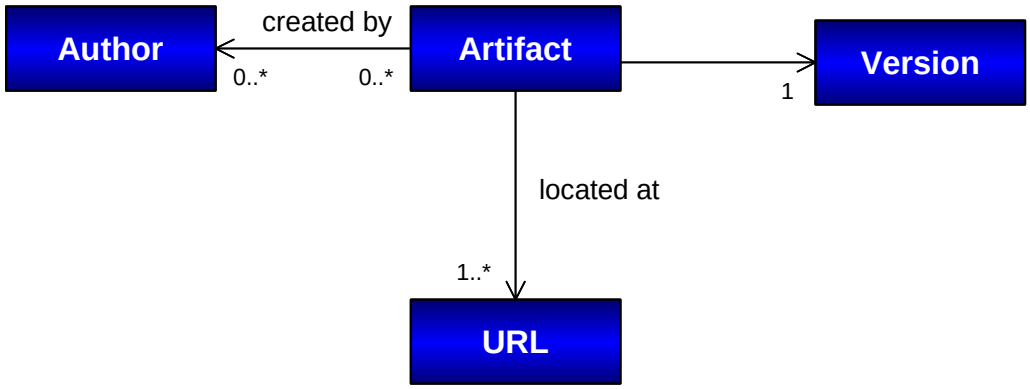

Figure 4: Artifact Associations

AKA, Keyword

An *AKA* represents another name for a pattern, strategy or component, while a *keyword* is a single word or phrase that serves to classify a pattern, strategy or component.

Both of these elements are defined by the attributes and associations enumerated in Table 12 and Table 13 respectively.

Table 12: AKA, Keyword Attributes

Value	Type	Required	Description
name	String	Yes	The alternate name or keyword.

Table 13: AKA, Keyword Associations

Element	Cardinality	Required	Description
Component	1	Yes	Component that is being classified or given another name
Pattern	1	Yes	Pattern that is being classified or given another name
Strategy	1	Yes	Strategy that is being classified or given another name

A graphic representation of the relationship of an AKA and keyword to other elements is provided in Figure 5.

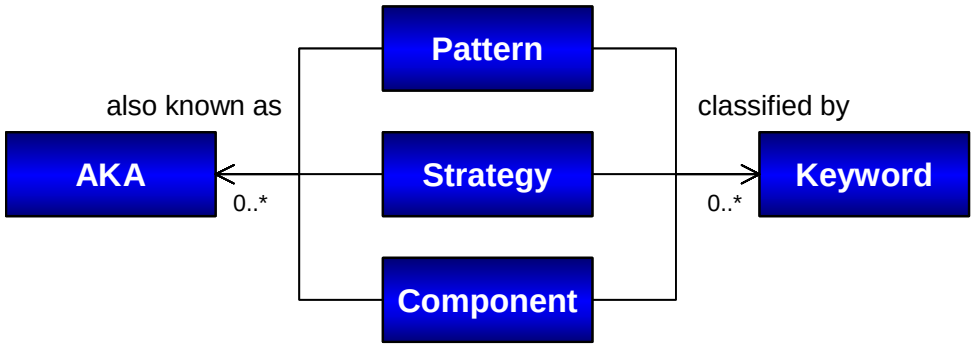

Figure 5: AKA, Keyword Associations

Chapter 4

Pattern Elements

Provides a detailed specification of patterns

Contents

Pattern

A *pattern* is a somewhat generic description of a solution provided to address one or a common set of problems in a certain context. Although a pattern describes a solution, it does not put any constraints on how that solution may be realized. A pattern may; however, describe how it relates to other patterns and even how it may be composed of other patterns. In this way, the abstract nature of patterns is preserved while the realization of solutions and idioms is reserved for strategies.

A pattern is defined by the attributes and associations enumerated in Table 14 and Table 15 respectively.

Table 14: Pattern Attributes

Value	Type	Required	Default	Description
namespace	String	Yes	--	A space within which the pattern name must be unique
name	String	Yes	--	Name of the pattern
abstraction	String	No	--	Abstraction level of the pattern, which may include such descriptions as “Architectural" or "Design"
domain	String	No	--	Domain the pattern is particularly well suited for or intended for, which may include such descriptions as "Financial," "Telecommunication," "Medical," etc.

Table 15: Pattern Associations

Element	Cardinality	Required	Description
AKA	0..*	No	Another name for the pattern
Artifact	0..*	No	An external resource that further describes the pattern
Author	0..*	No	An author of the pattern
Catalog	0..*	No	Organizes the pattern among others
Consequence	0..*	No	A consequences of the pattern's use

Element	Cardinality	Required	Description
Context	1	Yes	Environment of the pattern
Force	1..*	Yes	A motivation of the pattern
Keyword	0..*	No	A categorization or classification of the pattern
Pattern	0..*	No	A related pattern
Problem	1	Yes	Problem solved by the pattern
Solution	1	Yes	Solution to the problem provided by the pattern
Strategy	0..*	No	An implementation of the pattern solution
Version	1	Yes	Version information for the pattern

A graphic representation of a pattern's association with other elements is provided in Figure 6 and Figure 7.

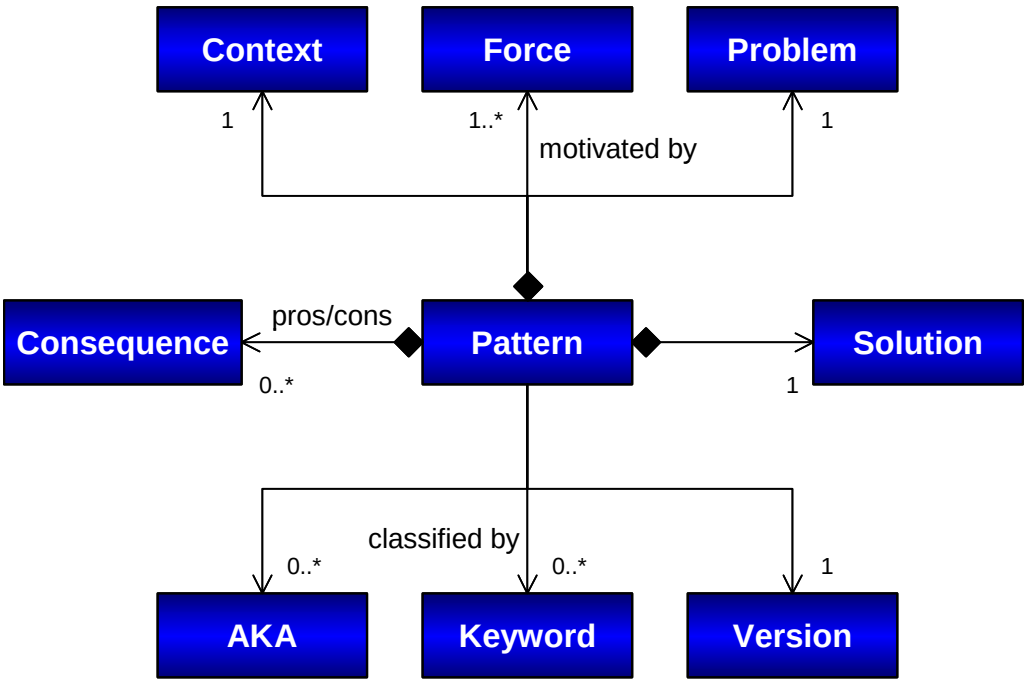

Figure 6: Pattern Associations, Part 1

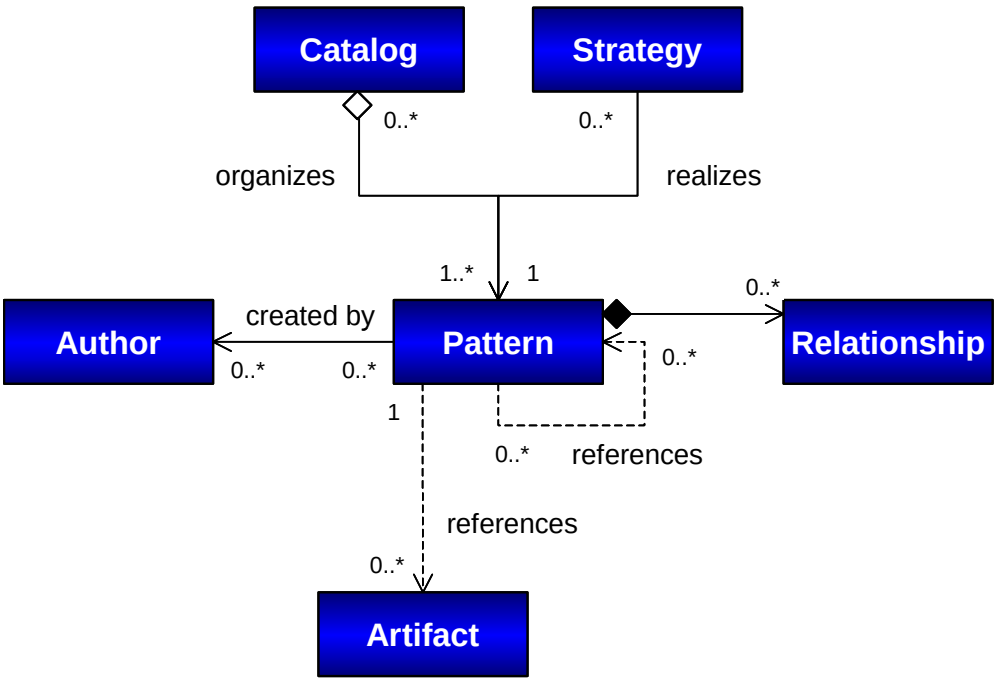

Figure 7: Pattern Associations, Part 2

Consequence, Context, Force, Problem

A *consequence* represents a pro or con of pattern usage. It describes how a pattern supports its objectives and the trade-offs in doing so.

A *context* represents the environment within which a pattern describes itself and is a general motivation for its existence.

A *force* represents a motivation of a pattern. It essentially amplifies the problem a pattern is trying to address and then serves as a constraint on the solution.

A *problem* represents a design need that is to be addressed by a pattern. It essentially distinguishes the use of one pattern over another.

All four of these elements are defined by the attributes and associations enumerated in Table 16 and Table 17 respectively.

Table 16: Consequence, Context, Force, and Problem Attributes

Value	Type	Required	Default	Description
description	String	Yes	--	Description of the force, problem or consequence
summary	String	No	--	A title or brief overview of the description

Table 17: Consequence, Context, Force, and Problem Associations

Element	Cardinality	Required	Description
Pattern	1	Yes	Parent pattern

A graphic representation of these elements and their association with others is provided in Figure 6.

Solution

A *solution* solves the problem described in a pattern. It is composed of a number of participants and defines the static structure and dynamic interactions of them.

A solution is defined by the attributes and associations enumerated in Table 18 and Table 19 respectively.

Table 18: Solution Attributes

Value	Type	Required	Default	Description
description	String	Yes	--	Description of the solution
summary	String	No	--	A title or brief overview of the description

Table 19: Solution Associations

Element	Cardinality	Required	Description

Element	Cardinality	Required	Description
Collaboration	1	Yes	Dynamic interactions found in the solution
Participant	0..*	No	A distinct role played by a component in the solution
Structure	1	Yes	Static structure of the solution

A graphic representation of a solution's association with other elements is provided in Figure 8.

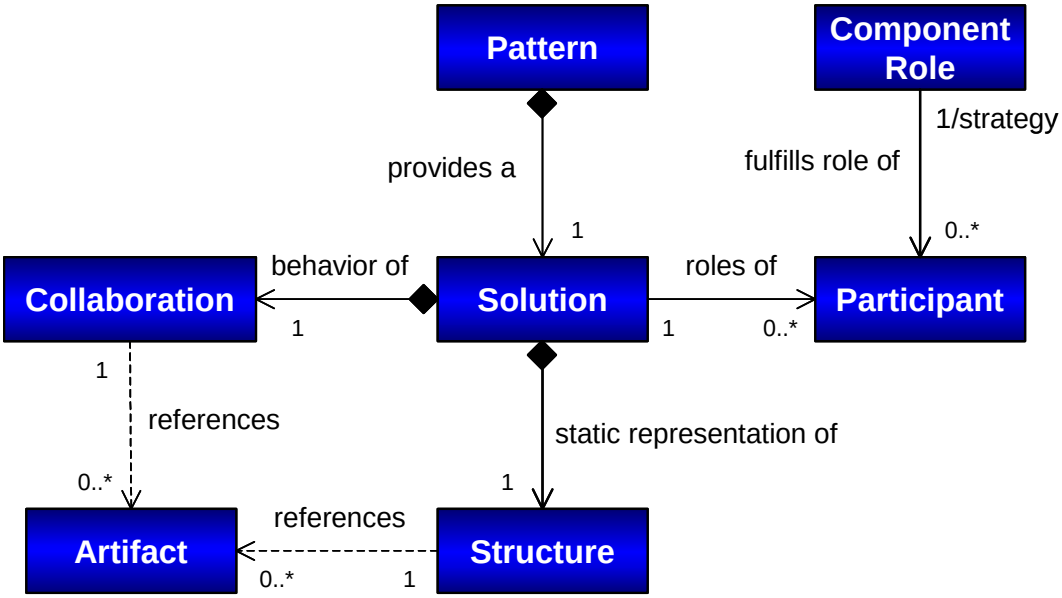

Figure 8: Solution Associations

Participant

A *participant* represents a distinct role played by a component in the pattern solution. Each participant describes its general characteristics but does not place any constraints on how it may be realized.

A participant is defined by the attributes and associations enumerated in Table 20 and Table 21 respectively.

Table 20: Participant Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the participant, which must be unique among the others
description	String	No	--	Description of the participant and its role in the solution
required	Boolean	No	true	Determines whether or not this participant is required to complete the solution

Table 21: Participant Associations

Element	Cardinality	Required	Description
Component Role	0..*	No	A component role in a pattern strategy that fulfills the role of the participant in the pattern solution
Solution	1	Yes	Parent solution that the structure or behavior describes

A graphic representation of a participant's association with other elements is provided in Figure 8.

Structure, Collaboration

A *structure* represents the static interaction of participants (as in a UML class diagram), while a *collaboration* represents the dynamic interaction of participants (as in a UML sequence or collaboration diagram).

Both are defined by the attributes and associations enumerated in Table 22 and Table 23 respectively.

Table 22: Structure, Collaboration Attributes

Value	Type	Required	Default	Description
description	String	Yes	--	Description of the structure or collaboration

Table 23: Structure, Collaboration Associations

Element	Cardinality	Required	Description
Artifact	0..*	No	An external artifact that may be referenced to further the description (i.e. UML diagrams).
Solution	1	Yes	Parent solution that the structure or collaboration describes

A graphic representation of the relationship of a structure and collaboration to other elements is provided in Figure 8.

Relationship

A *relationship* represents a relationship between two patterns. A pattern relationship is purely descriptive, but it does have an attribute that specifies what type of relationship it is. This element would be used to refer to a like pattern or to describe a pattern nesting.

A relationship is defined by the attributes and associations enumerated in Table 20 and Table 21 respectively.

Table 24: Relationship Attributes

Value	Type	Required	Default	Description
summary	String	Yes	--	A short phrase that describes the related pattern
description	String	No	--	Description of how the two patterns are related
type	String	No	“reference”	Defines the type of relationship. The following values are possible: ▪ like – both patterns are similar in one way or another ▪ nest – related pattern is nested in this one ▪ reference – a simple reference

Value	Type	Required	Default	Description
				to another pattern

Table 25: Relationship Associations

Element	Cardinality	Required	Description
Pattern	0..*	No	Parent pattern

A graphic representation of a relationship's association with other elements is provided in Figure 8.

XML Bindings

Each pattern is represented with an XML descriptor that has the ".pattern" file extension. The DTD for this descriptor is provided in *Pattern and Component Descriptors*. Files based on this DTD will typically be placed in a catalog JAR. Packaging requirements for this descriptor are discussed in more detail in *Packaging Requirements*.

Chapter 5

Strategy Elements

Provides a detailed specification of pattern strategies

Contents

Strategy

A strategy represents one of many possible implementations of a pattern solution, a building block for other strategies or an idiom. It serves as a bridge from the more abstract notion of a pattern to the more rigid world of components. A strategy can describe the design of a single component or a large framework of components. A strategy is role based, and each role defines restrictions on any component or element that may fill it. It is this role-based mechanism that gives strategies their greatest value; reuse of a design (which the strategy codifies) is gained by plugging in different components and elements in each role.

A strategy is defined by the attributes and associations enumerated in Table 26 and Table 27 respectively.

Table 26: Strategy Attributes

Value	Type	Required	Default	Description
namespace	String	Yes	--	A space within which the strategy name must be unique
name	String	Yes	--	Name of the strategy
description	String	No	--	Description of the strategy

Table 27: Strategy Associations

Element	Cardinality	Required	Description
AKA	0..*	No	Another name for the strategy
Artifact	0..*	No	External resources that further describe the strategy
Author	0..*	No	An author of the strategy
Catalog	0..*	No	Organizes the strategy with other strategies and patterns
Component Role	0..*	No	A role filled by a component
Connector Role	0..*	No	A role filled by a relationship between two components

Element	Cardinality	Required	Description
Keyword	0..*	No	A categorization or classification of the strategy
Pattern	0..1	No	The pattern for which the strategy provides an implementation to its solution
Strategy	0..*	No	Another strategy from which this one is composed
Version	1	Yes	Version information for the strategy

A graphic representation of a strategy's association with other elements is provided in Figure 9 and Figure 10.

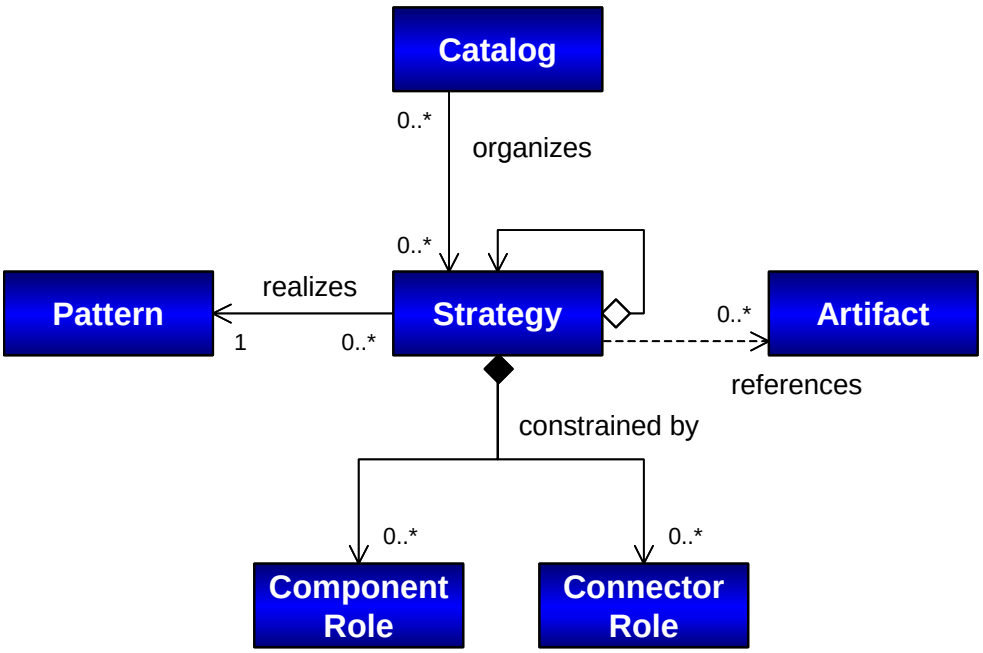

Figure 9: Strategy Associations, Part 1

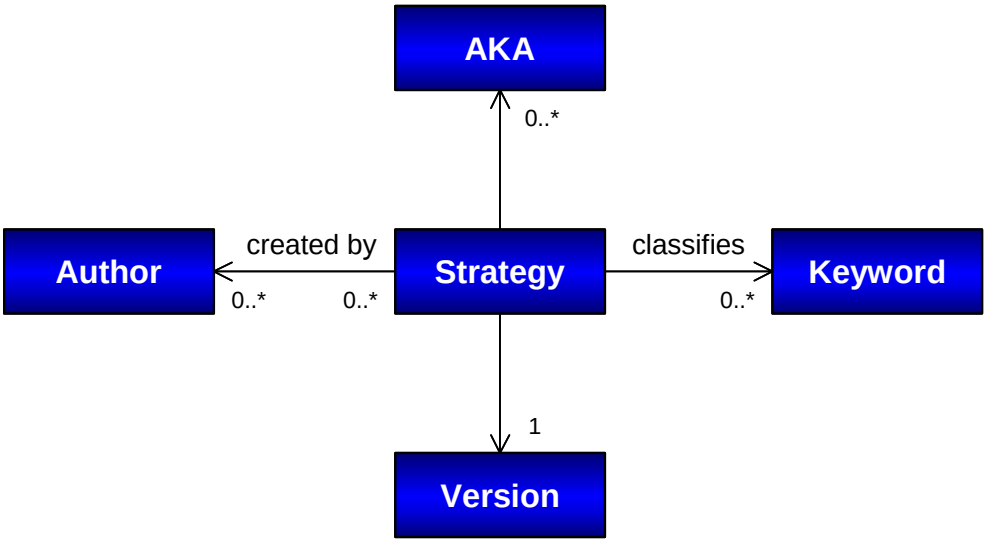

Figure 10: Strategy Associations, Part 2

Composite Strategies

A strategy may be composed of its own roles and any number of other strategies. The roles of the other strategies then become a part of the composite strategy. This mechanism provides the following benefits:

- Redundancy among strategies is removed. Duplicate roles among strategies may be factored out into a separate strategy.
- A component role from a nested strategy may be linked to a pattern participant that the composite strategy had not yet fulfilled. This is the mechanism through which pattern nesting is achieved.

Component Role

A *component role* represents a plug-in point in a strategy for a component. The role specifies an interface, so to speak, that a component must satisfy to fill the role. Any number of components may be swapped in and out of each component role, as long as they adhere to the specified interface.

A component role defined by the attributes and associations enumerated in Table 28 and Table 29 respectively.

Table 28: Component Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the component role
description	String	No	--	Description of the component role
stereotype	String	No	--	Defines the type of component the component role may be mapped to (see *UML Profiles* for more information)
is-interface	Boolean	No	false	Restricts the mapping of this component role to component interfaces only
multiplicity	String	No	1	Allows the role to be filled by more than one component. This is useful in patterns like Abstract Factory, where the concrete factory role will be mapped to multiple components. The following values are available: ▪ 1 - One ▪ # - Any whole number > 1 ▪ * - Many or more than one
required	Boolean	No	true	Determines whether or not this component role is required to be filled when a strategy is mapped

Table 29: Component Role Associations

Element	Cardinality	Required	Description
Attribute Role	0..*	No	Child attribute role
Component Role	0..1	No	An inheritance relationship with another component role. Both roles must have the same value for is-interface. If the component that fills this component role is composed of a single class, then it is required to subclass the component that fills the parent component role.

Element	Cardinality	Required	Description
Component Role	0..*	No	Interfaces that this component role implements. The associated component roles must have is-interface set to true.
Connector End Role	0..*	No	Component role participates in one end of a connector role
Operation Role	0..*	No	Child operation role
Participant	0..*	No	A pattern participant that is fulfilled by the component role
Strategy	1	Yes	Parent strategy
Tag Role	0..*	No	Child tag role

A graphic representation of a component role's association with other elements is provided in Figure 11 and Figure 12.

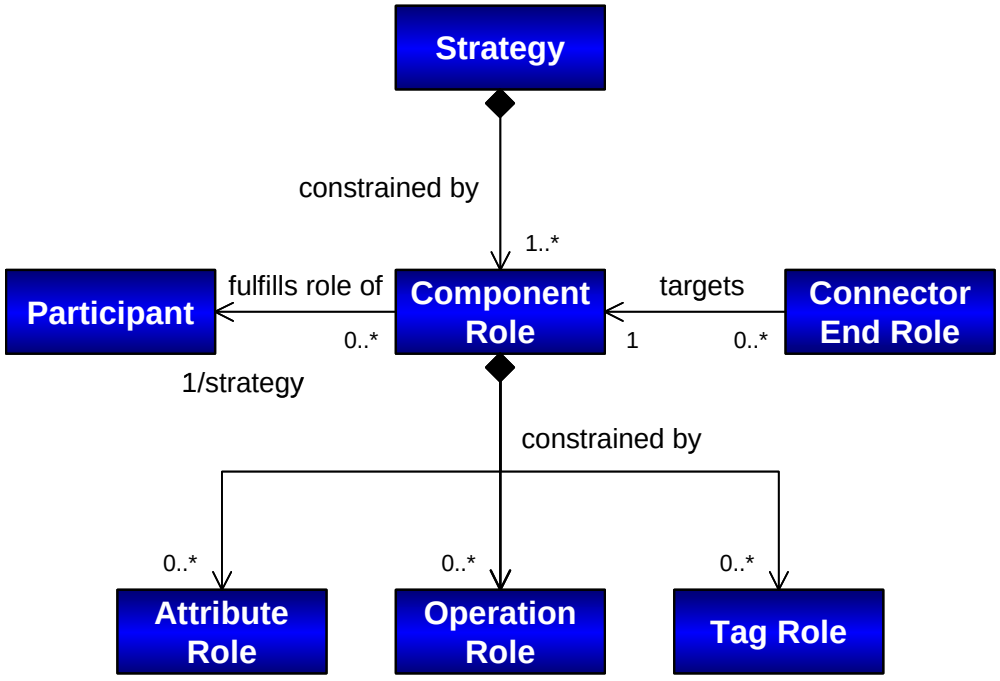

Figure 11: Component Role Associations, Part 1

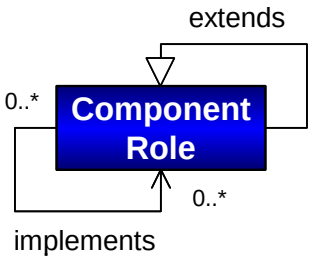

Figure 12: Component Role Associations, Part 2

Mapping Component Roles to Pattern Participants

If a strategy is providing an implementation for a pattern solution, then each pattern participant must be linked to a strategy component role. There may be more component roles in a strategy than there are pattern participants, so the converse is not true. This participant-role binding provides linkages between a pattern and one of its solution strategies. Not only does this mechanism show how a strategy relates to a pattern; it also shows how patterns nest (see *Composite Strategies*).

Attribute Role

An *attribute role* represents an attribute of a component. Each attribute role that is defined further restricts the components that a component role may be mapped to. Each required attribute role must be mapped to a valid attribute before the strategy is properly implemented.

An attribute role is defined by the attributes and associations enumerated in Table 30 and Table 31 respectively.

Table 30: Attribute Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the attribute role
type	String	No	--	Type that an attribute must be to satisfy the attribute role. Examples of Java types include “java.lang.String" and "boolean." The type may be represented using a

Value	Type	Required	Default	Description
				SCML substitution (see *SCML Extensions for Patterns* for more information).
description	String	No	--	Description of the attribute role
visibility	Access	No	public	Visibility that an attribute must have to satisfy the attribute role. For example, if the attribute role specifies a visibility of "public," then it may only be mapped to a public attribute.
static	Boolean	No	false	Determines whether or not the attribute that the attribute role is mapped to must belong to a component or an instance. For example, an attribute role with static set to "true" may not be mapped to an attribute that is owned by an instance.
constant	Boolean	No	false	Determines whether or not the attribute that the attribute role is mapped to must be a constant. For example, an attribute role with constant set to "true" may not be mapped to a mutable attribute.
multiplicity	String	No	1	Allows the role to be filled by more than one attribute. This is useful in patterns like Value Object, where the role representing data will be mapped to multiple attributes. The following values are available: ▪ 1 - One ▪ # - Any whole number > 1 ▪ * - Many or more than one
required	Boolean	No	true	Determines whether or not this attribute role is required to be mapped when its parent component role is mapped to a component

Table 31: Attribute Role Associations

Element	Cardinality	Required	Description

Element	Cardinality	Required	Description
Component Role	1	Yes	Parent component role.

A graphic representation of an attribute role's association with other elements is provided in Figure 11.

Operation Role

An *operation role* represents a method of a component. Each operation role that is defined further restricts the components that a component role may be mapped to. Each required operation role must be mapped to a valid method before the strategy is properly implemented.

An operation role is defined by the attributes and associations enumerated in Table 32 and Table 33 respectively.

Table 32: Operation Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the operation role
stereotype	String	No	--	Defines the type of method the operation role may be mapped to (see *UML Profiles* for more information)
description	String	No	--	Description of the operation role
body	String	No	--	Content of an operation role's body. It may contain source code, Source Code Macro Language (SCML), or a combination of both (see *SCML Extensions for Patterns* for more information).
visibility	Access	No	public	Visibility that a method must have to satisfy the operation role. For example, if the operation role specifies a visibility of "public", then it may only be mapped to a public method.
static	Boolean	No	false	Determines whether or not the method that the operation role is mapped to must belong to a component or an

Value	Type	Required	Default	Description
				instance. For example, an operation role with static set to "true" may not be mapped to a method that is owned by an instance.
return-type	String	No	--	Return type that a method must have to satisfy the operation role. For example, an operation role with a return type set to "boolean" may not be mapped to a method with a return type of "int" or one that has no return type. The return-type may be represented using an SCML substitution (see *SCML Extensions for Patterns* for more information).
multiplicity	String	No	1	Allows the role to be filled by more than one method. This is useful in patterns like Factory Method, where the actual factory method may be mapped to multiple methods. The following values are available: ▪ 1 - One ▪ # - Any whole number > 1 ▪ * - Many or more than one
required	Boolean	No	true	Determines whether or not this operation role is required to be mapped when its parent component role is mapped to a component

Table 33: Operation Role Associations

Element	Cardinality	Required	Description
Component Role	1	Yes	Parent component role
Parameter Role	0..*	No	Arguments that define part of the operation role’s signature. The list of parameter roles may be ordered or unordered.

A graphic representation of an operation role's association with other elements is provided in Figure 13.

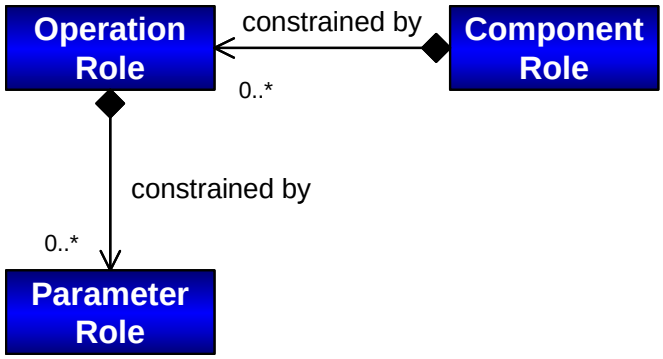

Figure 13: Operation Role Associations

Parameter Role

A *parameter role* represents a parameter of a method. Each parameter role that is defined further restricts the methods that the operation role may be mapped to. Each parameter role must be mapped to a valid parameter before the strategy is properly implemented.

A parameter role is defined by the attributes and associations enumerated in Table 34 and Table 35 respectively.

Table 34: Parameter Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the parameter role
type	String	Yes	--	Type of the parameter role. Examples of Java types include "boolean" and "java.lang.String." The return-type may be represented using an SCML substitution (see *SCML Extensions for Patterns* for more information).
constant	Boolean	No	false	Determines whether or not the parameter that the parameter role is mapped to must be a constant. For example, a parameter role with constant set to "true" may not be

Value	Type	Required	Default	Description
				mapped to a mutable parameter.
description	String	No	--	Description of the parameter role

Table 35: Parameter Role Associations

Element	Cardinality	Required	Description
Operation Role	1	Yes	Parent operation role

A graphic representation of a parameter role's association with other elements is provided in Figure 13.

Tag Role

A *tag role* represents a tag in a markup component (e.g. HTML, JSP). Each tag role that is defined further restricts the components that the component role may be mapped to. Each tag role must be mapped to a valid tag before the strategy is properly implemented.

A tag role is defined by the attributes and associations enumerated in Table 36 and Table 37 respectively.

Table 36: Tag Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the tag role
description	String	No	--	Description of the tag role
stereotype	String	No	--	Defines the type of tag the tag role may be mapped to (see *UML Profiles* for more information)
prefix	String	No	--	Default tag library prefix (for JSPs) or a namespace
tag-name	String	No	--	Literal name of the tag
multiplicity	String	No	1	Allows the role to be filled by more than one tag

Value	Type	Required	Default	Description
				than one tag The following values are available: ▪ 1 - One ▪ # - Any whole number > 1 ▪ * - Many or more than one
required	Boolean	No	true	Determines whether or not this tag role is required to be mapped when its parent component role is mapped to a component

Table 37: Tag Role Associations

Element	Cardinality	Required	Description
Component Role	1	Yes	Parent component role
Tag Attribute	0..*	No	An attributes of the tag
Tag Role	0..*	No	Nested tag roles

A graphic representation of a tag role's association with other elements is provided in Figure 14.

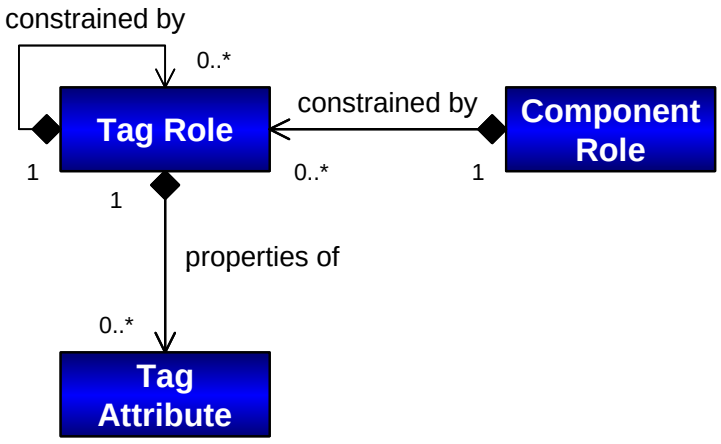

Figure 14: Tag Role Associations

Tag Attribute Role

A *tag attribute role* represents a markup tag attribute. Each tag attribute role that is defined further restricts the markup tags that the tag role may be mapped to. Each tag attribute role must be mapped to a valid tag attribute before the strategy is properly implemented.

A tag attribute is defined by the attributes and associations enumerated in Table 38 and Table 39 respectively.

Table 38: Tag Attribute Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the tag attribute
value	String	Yes	--	Value of the tag attribute
constant	Boolean	No	false	Determines whether or not the tag attribute that the tag attribute role is mapped to must be a constant. For example, a tag attribute role with constant set to "true" may not be mapped to a mutable tag attribute.
description	String	No	--	Description of the tag attribute role

Table 39: Tag Attribute Associations

Element	Cardinality	Required	Description
Tag Role	1	Yes	Parent tag role

A graphic representation of a tag attribute's association with other elements is provided in Figure 14.

Connector Role

A *connector role* represents a binary relationship between components. Each connector role has two end roles that must both be attached to a component role. Each connector role must be mapped to a valid relationship before the strategy is properly implemented.

A connector role is defined by the attributes and associations enumerated in Table 40 and Table 41 respectively.

Table 40: Connector Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the connector role.
description	String	No	--	Description of the connector role.
required	Boolean	No	true	Determines whether or not this connector role is required to be filled when a strategy is mapped

Table 41: Connector Role Associations

Element	Cardinality	Required	Description
Connector End Role	2	Yes	Connector ends
Pattern Strategy	1	Yes	Parent strategy

A graphic representation of a connector role's association with other elements is provided in Figure 15.

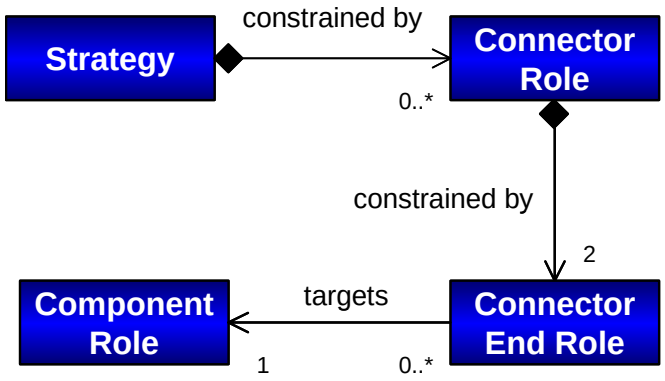

Figure 15: Connector Role Associations

Connector End Role

A *connector end role* represents one end of a binary relationship between components. Each connector end role further restricts which relationship a connector role may be mapped to.

A connector end role is defined by the attributes and associations enumerated in Table 42 and Table 43 respectively.

Table 42: Connector End Role Attributes

Value	Type	Required	Default	Description
name	String	Yes	--	Name of the connector end role
description	String	No	--	Description of the connector end role
multiplicity	String	Yes	--	Defines the required number of component roles for this end of the connector. The following multiplicities are allowed: ▪ 1 - One ▪ # - Any whole number ▪ 0..1 - Zero or One ▪ 0..* - Zero to Many ▪ 1..* - One to Many ▪ #..# - Any whole number range ▪ * - Many
navigable	Boolean	No	false	Determines whether or not this end is visible to the other
aggregation	Aggregation	No	aggregation	Defines the nature of this end role's association with the other one
changeability	Mutability	No	read-write	Defines the mutability of the end role (not of the component role that fills it)
visibility	Access	No	public	Defines the access other roles have to this connector end

Table 43: Connector End Role Associations

Element	Cardinality	Required	Description
Component Role	1	Yes	Component role that is the target of the connector end
Connector Role	1	Yes	Parent connector

A graphic representation of a connector end role's association with other elements is provided in Figure 15.

XML Bindings

Each strategy is represented with an XML descriptor that has the ".strategy" file extension. The DTD for this descriptor is provided in *Pattern and Component Descriptors*. Files based on this DTD will typically be placed in a catalog JAR. Packaging requirements for this descriptor are discussed in more detail in *Packaging Requirements*.

Chapter 6

Catalog Elements

Provides a detailed specification of pattern catalogs

Contents

Catalog

A *catalog* groups a number of related patterns and strategies according to some criteria. There is no restriction on how they are grouped, so it could be by domain, company, abstraction level, etc. A catalog serves as the basis for packaging and exchanging patterns and strategies. The physical structure of a catalog is provided in *Packaging Requirements*.

A catalog is defined by the attributes and associations enumerated in Table 44 and Table 45 respectively.

Table 44: Catalog Attributes

Value	Type	Required	Default	Description
namespace	String	Yes	--	A space within which the catalog name must be unique
name	String	Yes	--	Name of the catalog
description	String	No	--	A description of the catalog

Table 45: Catalog Associations

Element	Cardinality	Required	Description
Artifact	0..*	No	External resources that further describe the catalog
Author	0..*	No	An author of the catalog
Catalog	0..*	No	A catalog may be composed of other catalogs
Pattern	0..*	No	Reference to a pattern that is provided in the catalog
Strategy	0..*	No	Reference to a strategy that is provided in the catalog
Version	1	Yes	Version information for the catalog

A graphic representation of a catalog's association with other elements is provided in Figure 16.

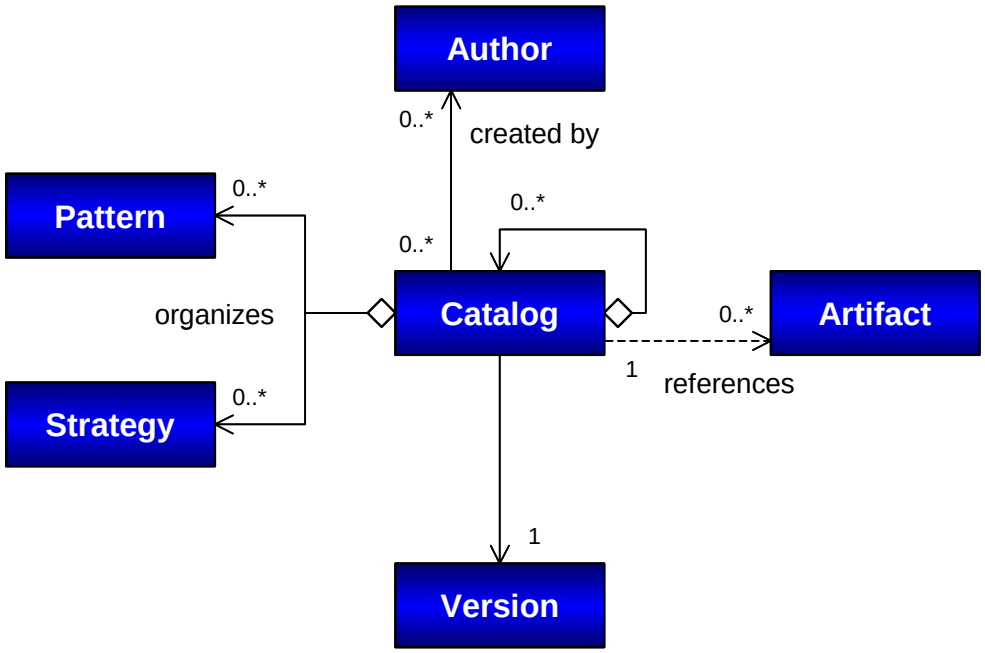

Figure 16: Catalog Associations

XML Bindings

Each catalog is represented with an XML descriptor that has the ".catalog" file extension. The DTD for this descriptor is provided in *Pattern and Component Descriptors*. Files based on this DTD will typically be placed in a catalog JAR. Packaging requirements for this descriptor are discussed in more detail in *Packaging Requirements*.

Chapter 7

SCML Extensions for Patterns

Provides SCML tags used specifically for patterns

Contents

Role References

The roles defined by a strategy often have the need to refer to one another. For example, an attribute role of one component role may have its type set to that of another component role. This is a difficult matter; since we cannot possibly know what components will ultimately fill the roles of a strategy. What we need, then, is a way for one role to reference another without loosing the flexibility that patterns and strategies provide.

The Source Code Macro Language (SCML) provides such a solution via its dynamic substitution mechanism. SCML is an XML-based macro language designed for source code generation. Source code is created via a number of SCML macros, which consist of source code mixed with SCML tags. Many of these tags are placeholders for substitutions that are made when a client requests that the macros be expanded. The substitutions are made via reflection from a graph of property-based objects that the client provides.

The substitution (or placeholder) mechanism is of particular interest here. In our example above, the attribute role would simply use a placeholder for its type that references the component role. When the strategy is later mapped to concrete components, the type of the component that fills the component role will be substituted (when mapping to existing components) or required (when using the strategy as a component creation template) for the attribute that fills the attribute role.

This section will define what the substitution mechanism for strategies looks like and how it can be used. The actual substitution process that must occur during the mapping of roles takes place via an SCML code generation implementation, as defined in that specification. We will use the `<s>` tag here for referencing one role and its properties from another and modify its syntax as necessary.

Modified <s> Tag

Syntax

```
<scml:s role="value">
  (strategy element)[/property]
</scml:s>
```

Where `role` must be a value of `true` or `false`. A value of `true` is the default and results in the property being resolved from the pattern strategy itself. A value of `false` results in the property being resolved from the component or component element that is mapped to the strategy. The `(strategy element)` placeholder will be explained in detail in the following sections.

Component Role References

Syntax

`<scml:s>component:`*`rolename`*`[/`*`property`*`]</scml:s>`

or

`<scml:s>c:`*`rolename`*`[/`*`property`*`]</scml:s>`

Where `rolename` is the name of the component role being referenced, and `property` is the component role property being referenced. Any component role property, as defined in this specification, may be accessed. If no property is specified, the component role `name` property will be used.

Example

Let's say we are filling in the body of an operation role. We have the need of referring to a component role, named `Account`, to do some type casting from an object that we are getting out of a collection. The statement looks like this:

```
(<scml:s role="false">c:Account</scml:s>) account =
(<scml:s role="false">c:Account</scml:s>)it.next();
```

When the pattern strategy is mapped, the component that is mapped to the `Account` component role will have its name substituted. If the name of the mapped component were `MyAccount`, the resulting statement in the method that the operation role is mapped to would look like this:

```
(MyAccount) account = (MyAccount)it.next();
```

Constraints

- The properties of a component role (name, stereotype, etc.) may be referenced anywhere within an operation role body.
- The `name` property of a component role may be referenced from:

- The `type` of another attribute role
- The `type` of a parameter role from an operation role
- The `return-type` of an operation role

Attribute Role References

Syntax

`<scml:s>[component:`*`rolename`*`;]attribute:`*`rolename`*`[/`*`property`*`]</scml:s>`

or

`<scml:s>[c:`*`rolename`*`;]a:`*`rolename`*`[/`*`property`*`]</scml:s>`

Where `rolename` is the name of the attribute role being referenced, and `property` is the attribute role property being referenced. Any attribute role property, as defined in this specification, may be accessed. If no property is specified, the attribute role `name` property will be used. When an attribute role is being referenced within the same component role, the prepended component role name is optional.

Example

For our first example, we have one component role that specifies two attribute roles that are named “frik” and “frak” respectively. The frik role has a type of java.lang.String. Since the frak attribute will always have the same type as frik, we set his type using the following placeholder:

`<scml:s role=”false”>attribute:frik/type<scml:s>`

or

`<scml:s role=”false”>a:frik/type<scml:s>`

When the component role is mapped to a concrete component, the type of the attribute that fills the frik role will be required for the attribute that fills the frak role. Since both attributes are within the same component role, there is no need to specify which component role frik belongs to within the `<s>` tag.

For our second example, we have the same two attribute roles, but now they each belong to a different component role: “componentA” and “componentB” respectively. The same reference would then use the following placeholder:

```
<scml:s role="false">
  component:componentA;attribute:frik/type
<scml:s>
```

or

```
<scml:s role="false">c:componentA;a:frik/type<scml:s>
```

Since both attributes are no longer within the same component role, the component role frik belongs to must be specified within the `<s>` tag.

Constraints

- The properties of an attribute role (name, type, etc.) may be referenced anywhere within an operation role body.
- The `type` property of an attribute role may be referenced from:
  - The `type` of another attribute role
  - The `type` of a parameter role from an operation role
  - The `return-type` of an operation role

Operation Role References

Syntax

```
<scml:s>[component:rolename;]operation:rolename[/property]
</scml:s>
```

or

```
<scml:s>[c:rolename;]o:rolename[/property]</scml:s>
```

Where `rolename` is the name of the operation role being referenced, and `property` is the operation role property being referenced. Any operation role property, as defined in this specification, may be accessed. If no property is specified, the operation role `name` property will be used.

Example

Let's say we are filling in the body of an operation role. We have the need of referring to another operation role contained within the same component role. The name of this other role is `doSomething`. The statement looks like this:

```
<scml:s role="false">o:doSomething</scml:s>();
```

When the pattern strategy is mapped, the method that is mapped to the `doSomething` operation role will have its name substituted. If the name of the mapped method were `processPayment`, the resulting statement in the method that the original operation role is mapped to would look like this:

```
processPayment();
```

Constraints

- The properties of an operation role (name, stereotype, etc.) may be referenced anywhere within an operation role body.

Parameter Role References

Syntax

<scml:s>parameter:*rolename*[*/property*]</scml:s>

or

<scml:s>p:*rolename*[*/property*]</scml:s>

Where `rolename` is the name of the parameter role being referenced, and `property` is the parameter role property being referenced. Any parameter role property, as defined in this specification, may be accessed. If no property is specified, the parameter role `name` property will be used.

Example

Let's say we simply wish to print out the value of a parameter that is mapped to a parameter role named `id`. The statement looks like this:

```
System.out.println(<scml:s role="false">p:id</scml:s>);
```

When the pattern strategy is mapped, the parameter that is mapped to the `id` parameter role will have its name substituted. If the name of the mapped parameter were `studentId`, the resulting statement in the method that the operation role is mapped to would look like this:

```
System.out.println(studentId);
```

Constraints

- The properties of a parameter role (name, type, etc.) may be referenced anywhere within an operation role body.
- Parameter roles may only be referenced within the body of the operational role in which they are defined.

Tag Role References

Syntax

`<scml:s>tag:`*`rolename`*`[`*`/property`*`]</scml:s>`

or

`<scml:s>t:`*`rolename`*`[`*`/property`*`]</scml:s>`

Where `rolename` is the name of the tag role being referenced, and `property` is the tag role property being referenced. Any tag role property, as defined in this specification, may be accessed. If no property is specified, the tag role `name` property will be used.

Example

`TODO`

Constraints

- The properties of a tag role (name, type, etc.) may be referenced anywhere within a tag role body.

Tag Attribute Role References

Syntax

`<scml:s>tagAttribute:`*`rolename`*`[/`*`property`*`]</scml:s>`

or

`<scml:s>ta:`*`rolename`*`[/`*`property`*`]</scml:s>`

Where `rolename` is the name of the tag attribute role being referenced, and `property` is the tag attribute role property being referenced. Any tag attribute role property, as defined in this specification, may be accessed. If no property is specified, the tag attribute role `name` property will be used.

Example

`TODO`

Constraints

- The properties of a tag attribute role (name, type, etc.) may be referenced anywhere within a tag role body.
- Tag attribute roles may only be referenced within the body of the tag role in which they are defined.

Connector Role References

Syntax

`<scml:s>connector:`*`rolename`*`[/`*`property`*`]</scml:s>`

or

`<scml:s>x:`*`rolename`*`[/`*`property`*`]</scml:s>`

Where `rolename` is the name of the connector role being referenced, and `property` is the connector role property being referenced. Any connector role property, as defined in this specification, may be accessed. If no property is specified, the connector role `name` property will be used.

Constraints

- The properties of a connector role (name, etc.) may be referenced anywhere within an operation role body.

Connector End Role References

Syntax

`<scml:s>[component:`*`rolename`*`;]connectorEnd:`*`rolename`*`[/`*`property`*`]</scmls>`

or

`<scml:s>[c:`*`rolename`*`;]xe:`*`rolename`*`[/`*`property`*`]</scml:s>`

Where `rolename` is the name of the connector end role being referenced, and `property` is the connector end role property being referenced. Any connector end role property, as defined in this specification, may be accessed. If no property is specified, the connector end role `name` property will be used.

Example

Let's say we are filling in the body of an operation role. We have the need of referring to a connector end role, named `Holdings`, contained within the same component role. The statement looks like this:

```
Collection holdings =

  get<scml:s role="false">xe:Holdings</scml:s>();
```

When the pattern strategy is mapped, the connector end that is mapped to the `Holdings` connector end role will have its name substituted. If the name of the mapped connector end were `StockHoldings`, the resulting statement in the method that the operation role is mapped to would look like this:

```
Collection holdings = getStockHoldings();
```

Constraints

- The properties of a connector end role (name, multiplicity, etc.) may be referenced anywhere within an operation role body.

Collections of Roles

Until now, we have only spoken of referencing single-valued properties of pattern strategy roles. It is possible to access, say, a collection of attribute roles from a particular component role. What you would do with such a collection will be discussed in the next section. This section will describe what collections are available, how to access them and where they may be used.

The collections that are enumerated below have actually already been defined in the pattern metamodel. We are now defining a syntax for accessing them.

Modified <for> Tag

Syntax

```
<scml:for var="var" property="property" role="value">
  ...
</scml:for>
```

The only change here to the SCML <for> tag is the addition of the `role` attribute, which is explained in the *Modified <s> Tag* section.

Common Constraints

- Collections are only accessible inside an operation role body as part of an SCML `<for>` tag.

Strategy Collections

A strategy is composed of any number of component roles and connector roles. Both of these collections are available through the following SCML substitutions.

Component Role Collection

```
<scml:for var="var"
property="strategy:this/componentRoles">
  ...
</scml:for>
```

Connector Role Collection

```
<scml:for var="var" property="s:this/connectorRoles">
  ...
</scml:for>
```

These substitutions produce a collection of component or connector roles from within the pattern strategy they are accessed from. Inside the `<for>` tag, the current component or connector role can be accessed through the `var` attribute.

Connector Role Collections

A connector role is composed of two connector end roles. These collections are available through the following SCML substitution:

```
<scml:for var="var"
     property="connector:name/connectorEndRoles">
  ...
</scml:for>
```

or

```
<scml:for var="var"
     property="x:name/connectorEndRoles">
  ...
</scml:for>
```

These substitutions produce a collection of connector end roles. Inside the `<for>` tag, the current connector end role can be accessed through the `var` attribute. The target connector is specified by name via the `name` attribute.

Component Role Collections

A component role is composed of any number of operation roles, attribute roles and tag roles. These collections are available through the following SCML substitutions.

Operation Role Collection

```
<scml:for var="var"
     property="component:name/operationRoles">
  ...
</scml:for>
```

or

```
<scml:for var="var"
     property="c:name/operationRoles">
  ...
</scml:for>
```

Attribute Role Collection

```
<scml:for var="var"
     property="component:name/attributeRoles">
  ...
</scml:for>
```

or

```
<scml:for var="var"
     property="c:name/attributeRoles">
  ...
</scml:for>
```

Tag Role Collection

```
<scml:for var="var"
     property="component:name/tagRoles">
  ...
```

```
</scml:for>
```

or

```
<scml:for var="var"
     property="c:name/tagRoles">
  ...
</scml:for>
```

Connector End Role Collection

```
<scml:for var="var"
     property="component:name/connectorEndRoles">
  ...
</scml:for>
```

or

```
<scml:for var="var"
     property="c:name/connectorEndRoles">
  ...
</scml:for>
```

These substitutions produce a collection of roles. Inside the `<for>` tag, the current role can be accessed through the `var` attribute.

Operation Role Collections

A component role is composed of any number of parameter roles. This collection is available through the following SCML substitution:

```
<scml:for var="var"
     property="operation:this/parameterRoles">
  ...
</scml:for>
```

or

```
<scml:for var=”var”
     property=”o:this/parameterRoles”>
  ...
</scml:for>
```

This substitution produces a collection of parameter roles. Inside the `<for>` tag, the current role can be accessed through the `var` attribute. This collection only returns the parameter roles for the current operation role.

Operation Role Bodies and SCML

Any operation role body may be a mixture of source code and SCML tags. When performing substitutions, the syntax defined above in the Role References section must be observed. This syntax replaces that which is defined for the `<s>` tag and its variants in the SCML specification.

All other SCML tags may also be used, and their use is governed by the SCML specification. This means that you may insert SCML macros (`<include>` tag), use conditional logic (`<if-equal>` tag), perform looping (`<for>` tag), etc. The only difference for using them with patterns is that all property references must adhere to the syntax defined in the *Role References* and *Collections of Roles* sections.

Example

Let’s say that we wish to loop through all of a component role’s (named `CompA`) operation roles and perform some steps based on conditional logic. The SCML might look like this:

```
<scml:for var=”role” property=”c:CompA/operationRoles”>
<scml:if-equal>
 <scml:value><scml:s>role/stereotype</scml:s></scml:value>
 <scml:value>EJBCreate</scml:value>
```

```
 <scml:then>
   System.out.println(“<scml:s>role/name</scml:s> is a
                      create method!”);
   ...
 </scml:then>
 <scml:else>
   System.out.println(“<scml:s>role/name</scml:s> is not a
                      create method!”);
   ...
 </scml:else>
</scml:if-equal>
```

Chapter 8

Component Elements

Provides a detailed specification of components

Contents

Component

A *component* represents an actual component. We do not describe a component's interface or internals here, because that is already done well through the particular component standard in use as well as UML itself. Instead, we describe authorship, versioning, external artifacts, etc. We also describe how a component and its elements fill strategy roles.

A component is defined by the attributes and associations enumerated in Table 46 and Table 47 respectively.

Table 46: Component Attributes

Value	Type	Required	Default	Description
namespace	String	Yes	--	A space within which the component name must be unique
name	String	Yes	--	Name of the component
description	String	No	--	A description of the component
type	String	No	--	Component type. This will be based on component stereotypes that are provided in *UML Profiles*

Table 47: Component Associations

Element	Cardinality	Required	Description
Artifact	0..*	No	An external resource that further describe the component
Author	0..*	No	An author of the component
Component Role	0..*	No	Component role filled by the component in a strategy
Keyword	0..*	No	Classifies the component
Palette	0..*	No	Organizes the component with other components
Strategy	0..*	No	A strategy the component role participates in

Element	Cardinality	Required	Description
URL	1..*	Yes	URL to a file, JAR or ZIP that defines part of or the entire component implementation
Version	1	Yes	Version information for the component

A graphic representation of a component's association to other elements is provided in Figure 17.

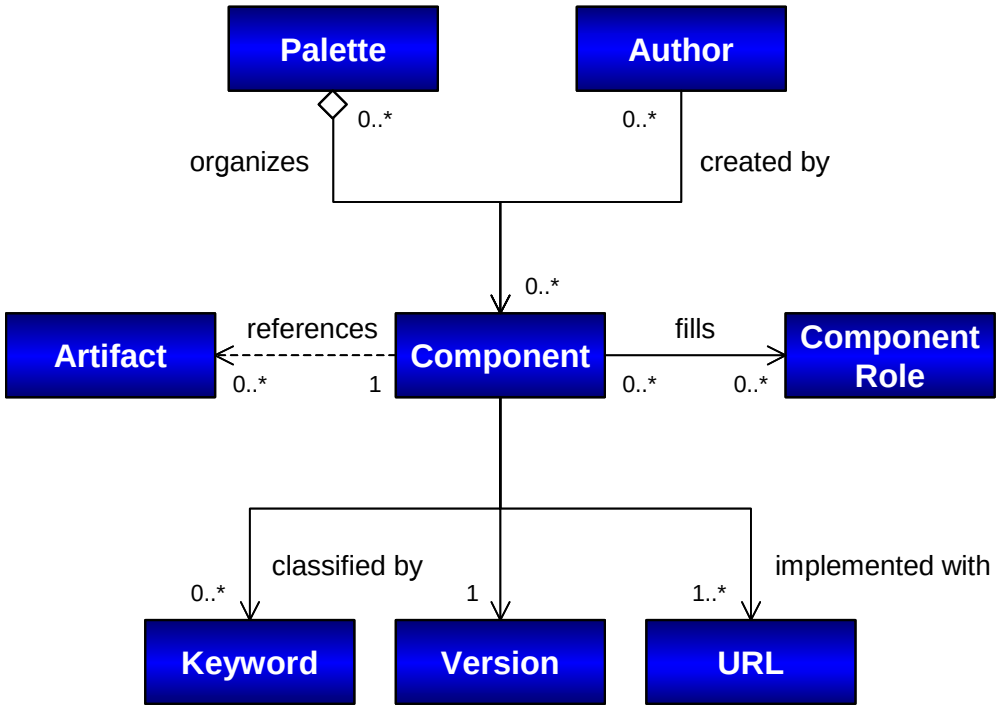

Figure 17: Component Associations

Mapping Roles to Components

When a strategy is instantiated, all required roles are mapped to one or more components or component elements (depending on the multiplicity). Roles that are not marked as required may or may not be mapped.

Each role serves to restrict which components may participate in a strategy. The more component and connector roles a strategy specifies, the more restrictive it is about which collaboration of components may represent it. Likewise, the more

attribute, operation and tag roles a component role specifies, the more restrictive it is about what component may fill it. Roles only serve to describe the required parts of a component that are necessary for its participation in the strategy, and they are not intended to describe an entire component. So a component that participates in an instantiated strategy will likely have some elements (attributes, operations, etc.) also participating as necessary to fill the associated component role and others that do not.

A pattern-driven development tool would restrict what components, methods, tags, attributes, etc. (as appropriate) are available for any given role in a strategy. If a component qualifies for filling a component role but does not have, say, an attribute that qualifies for filling a required attribute role, the tool could offer to generate the attribute on the component. In this way, strategies may be mapped onto existing components, used as templates to create a collaboration of components or a mixture of both.

XML Bindings

Each component is represented with an XML descriptor that has the ".component" file extension. The DTD for this descriptor is provided in *Pattern and Component Descriptors*. Files based on this DTD will typically be placed alongside a components implementation classes.

Each instantiated strategy is represented with an XML descriptor that has the ".istrategy" file extension. The DTD for this descriptor is provided in *Pattern and Component Descriptors*. Files based on this DTD will typically be packaged as part of a component palette; however, they may also be packaged with individual or a subset of components (e.g. an EJB JAR).

Packaging requirements for both of these descriptors are discussed in more detail in *Packaging Requirements*.

Chapter 9

Palette Elements

Provides a detailed specification of component palettes

Contents

Palette

A *palette* groups a number of related components according to some criteria. There is no restriction on how they are grouped, so it could be by domain, company, type, function, etc. A palette serves as the basis for packaging and exchanging a group of reusable components and frameworks. If instantiated strategies were included with the components or framework, then including catalogs containing the referenced patterns and strategies would not be uncommon. The physical structure of a palette is provided in *Packaging Requirements*.

A palette is defined by the attributes and associations enumerated in Table 48 and Table 49 respectively.

Table 48: Palette Associations

Value	Type	Required	Default	Description
namespace	String	Yes	--	A space within which the palette name must be unique
name	String	Yes	--	Name of the palette
description	String	No	--	A description of the palette

Table 49: Palette Associations

Element	Cardinality	Required	Description
Artifact	0..*	No	An external resource that further describes the palette
Author	0..*	No	An author of the palette
Catalog	0..*	No	A catalog that contains patterns and strategies pertinent to the component or framework design
Component	0..*	No	A component that is provided on the palette
Palette	0..*	No	A palette may be composed of other palettes
Version	1	Yes	Version information for the palette

A graphic representation of a palette's association with other elements is provided in Figure 18.

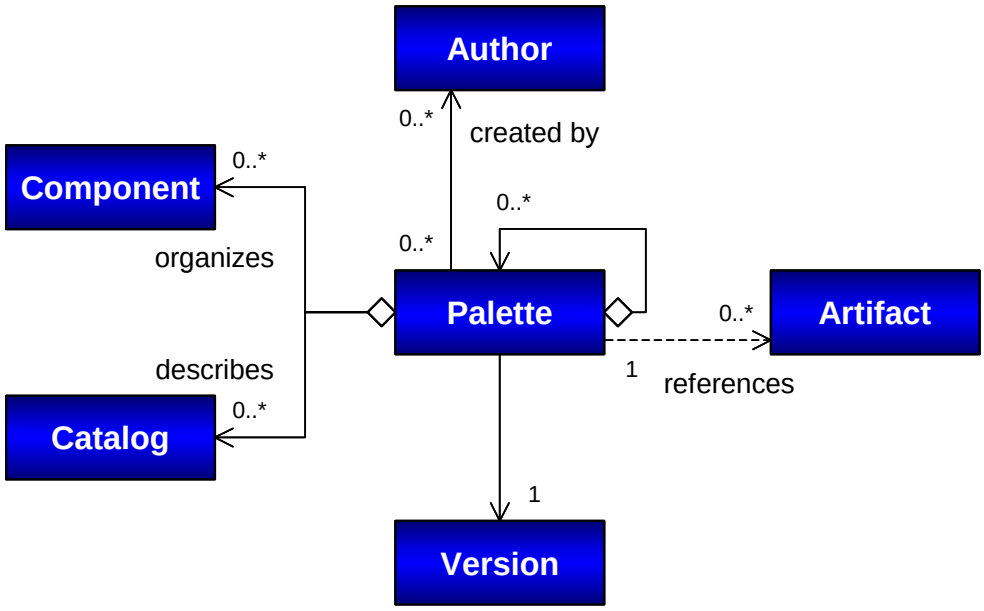

Figure 18: Palette Associations

XML Bindings

Each palette is represented with an XML descriptor that has the ".palette" file extension. The DTD for this descriptor is provided in *Pattern and Component Descriptors*. Files based on this DTD will typically be placed in a palette JAR. Packaging requirements for this descriptor are discussed in more detail in *Packaging Requirements*.

Chapter

10

UML Profiles

Provides a UML profile for patterns and defines which other profiles may be used for stereotypes

Contents

TODO

Chapter

11

Packaging Requirements

Provides requirements for packaging patterns and components

Contents

TODO

Chapter 12

Examples

Provides non-trivial examples of PCML in action

Contents

TODO

Chapter 13

Pattern and Component Descriptors

Provides XML DTDs for elements defined in this specification

Contents

Common Elements DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2001-2002 ObjectVenture Inc.  All rights
     reserved. This product or document is protected by
     copyright and distributed under licenses restricting its
     use, copying, and distribution.  No part of this product
     or documentation may be reproduced in any form by any
     means without prior written authorization of ObjectVenture
     and its licensors, if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining common constructs used in the
     description of patterns and components. Standalone XML
     files derived from this DTD are not recommended. The
     parent DTD of any element that wishes to include any of
     these constructs should reference this DTD as an external
     entity.

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- The Boolean entity is the string representation of a
     boolean (true or false) variable.
-->
<!ENTITY % Boolean "(true | false | yes | no)">


<!-- A location entity is optionally one of the following:
```

```
     1) a relative path, delimited by "/" characters, that
        defines the location of a resource relative to the
        location of the XML file it is referenced within
     2) or a URI path to an external resource.

-->
<!ENTITY % Location "CDATA">


<!-- The Access entity is the string representation of an
     element's visibility to others.
-->
<!ENTITY % Access "(public | private | protected)">


<!-- The Aggregation entity is the string representation of an
     element's level of aggregation over another in a
     relationship.
-->
<!ENTITY % Aggregation "(composition | aggregation | none)">


<!-- The Mutability entity is the string representation of an
     element's changeability.
-->
<!ENTITY % Mutability "(read | read-write)">


<!-- ========== Common Elements ========================== -->


<!-- A description element is an explanation of another
     element.
-->
<!ELEMENT description (#PCDATA)>


<!-- A summary element is a quick summary of another element.
-->
<!ELEMENT summary (#PCDATA)>


<!-- ========== URL Element ============================== -->


<!-- An urls element is a section that contains one or more
     url elements.

       url              A URL
-->
<!ELEMENT urls (url+)>


<!-- An url element is an URL link with a friendly display
```

```
     name.

       display-name   Display name of the URL

       address        Actual URL

     An example would be a display-name of "My Home Page" with
     an address of "www.johndoe.com."
-->
<!ELEMENT url EMPTY>
<!ATTLIST url          display-name   CDATA          #IMPLIED>
<!ATTLIST url          address        CDATA          #REQUIRED>


<!-- ========== AKA Element ================================ -->


<!-- An akas element is a section that contains one or more
     aka elements.

       aka              An aka
-->
<!ELEMENT akas (aka+)>


<!-- An aka element is an another name that the parent element
     may be known by.
-->
<!ELEMENT aka (#PCDATA)>


<!-- ========== Keyword Element ============================ -->


<!-- A keywords element is a section that contains one or more
     keywords.

       keyword         A keyword
-->
<!ELEMENT keywords (keyword+)>


<!-- A keyword element is word or phrase that is useful in
     categorizing the parent element and its characteristics.
-->
<!ELEMENT keyword (#PCDATA)>


<!-- ========== Author Element ============================= -->


<!-- An authors element is a section that contains one or more
     authors.
```

```
       author          An author
-->
<!ELEMENT authors (author+)>


<!-- An author element contains information identifying the
     creator of a resource.

     name           Name of the author

     organization   Organization the author represents. If the
                    name is an actual organization, then this
                    attribute may be omitted.

     description    Description of the author

     url            A URL where information pertaining to the
                    author, his organization, or his works may
                    be obtained. This includes e-mail
                    addresses.
-->
<!ELEMENT author (description, url*)>
<!ATTLIST author       name           CDATA          #REQUIRED>
<!ATTLIST author       organization   CDATA          #IMPLIED>


<!-- ========== Version Element ========================== -->


<!-- A version element represents versioning information of a
     resource. It's primary purpose is to distinguish multiple
     revisions of the same resource.

     revision       Version number

     date           Date/time of the revision

     description    Description of the revision

     copyright      Copyright notice for this revision of the
                    resource

     release-notes  Notes that describe important aspects of
                    this revision

     license        Licensing information for this revision of
                    the resource
-->
<!ELEMENT version (description, copyright?, release-notes?,
                   license?, artifacts?)>
<!ATTLIST version      revision       CDATA          #REQUIRED>
<!ATTLIST version      date           CDATA          #IMPLIED>
```

```
<!-- The copyright element provides a copyright notice.
-->
<!ELEMENT copyright (#PCDATA)>


<!-- The release-notes element provides a description of a
     resource revision.
-->
<!ELEMENT release-notes (#PCDATA)>


<!-- The license element provides licensing information for a
     resource that, among other things, defines usage
     restrictions.
-->
<!ELEMENT license (#PCDATA)>


<!-- ========== Artifact Element ========================== -->


<!-- An artifacts element is a section that contains one or
     more artifacts.

       artifact         An artifact
-->
<!ELEMENT artifacts (artifact+)>


<!-- An artifact element is an external file that may not be
     appropriately supplied in XML form. When related to
     patterns, it helps to further describe a pattern or
     instruct in its use. Examples of an artifact include: UML
     diagram, graphical image, binary documentation, etc.

     name           Name of the artifact

     type           File type of the artifact, which should be
                    represented by a common file extension
                    (i.e. html, doc, mdl)

     url            Location of the artifact in the form of a
                    URL, which may be either relative or
                    absolute

     description    Description of the artifact

     author         An author of the artifact

     version        Version information for the artifact
-->
<!ELEMENT artifact (description?, authors?, version, urls?)>
<!ATTLIST artifact     name           CDATA          #REQUIRED>
<!ATTLIST artifact     type           CDATA          #REQUIRED>
```

```
<!ATTLIST artifact     url            %Location;     #REQUIRED>
```

Pattern DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2001-2002 ObjectVenture Inc.  All rights
     reserved.

     This product or document is protected by copyright and
     distributed under licenses restricting its use, copying,
     and distribution. No part of this product or documentation
     may be reproduced in any form by any means without prior
     written authorization of ObjectVenture and its licensors,
     if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining a pattern. To support validation
     of your pattern file, include the following DOCTYPE
     element at the beginning (after the "xml" declaration):

     <!DOCTYPE pattern PUBLIC
       "-//ObjectVenture//DTD Pattern 1.0//EN"
       "http://www.objectventure.com/dtds/pattern-1_0.dtd">

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- This entity is a reference to an external DTD. It defines
     a number of common entity and element definitions that are
     used here and in the other pattern DTDs.
-->
<!ENTITY % common SYSTEM "common.dtd">
%common;
```

```
<!-- The RelType entity represents a type of relationship
     between two patterns.
-->
<!ENTITY % RelType "(like | nested | reference)">
```

<!-- ========== Pattern Element ========================== -->

<!-- A pattern is a somewhat generic description of a solution provided to address one or a common set of problems in a certain context. Although a pattern describes a solution, it does not put any constraints on how that solution may be realized. A pattern may; however, describe how it relates to other patterns and even how it may be composed of other patterns. In this way, the abstract nature of patterns is preserved while the realization of solutions and idioms is reserved for strategies.

namespace	A space within which the pattern name must be unique
name	Name of the pattern
abstraction	Abstraction level of the pattern, which may include such descriptions as “Architectural" or "Design”
domain	Domain the pattern is particularly well suited for or intended for, which may include such descriptions as “Financial,” “Telecommunication,” “Medical,” etc.
authors	Authors of the pattern
version	Version information for the pattern
akas	Other names for the pattern
keywords	Categorizations or classifications of the pattern
context	Environment of the pattern
forces	Motivation of the pattern
problem	The problem solved by the pattern
solution	The solution to the problem provided by the pattern
consequences	Consequence of the pattern's use

```
     relationships    Other related patterns

     artifacts        External resources that further describes
                      the pattern
-->
<!ELEMENT pattern (authors?, version, akas?, keywords?,
                   context, forces, problem, solution,
                   consequences, relationships?, artifacts?)>
<!ATTLIST pattern      namespace      CDATA          #REQUIRED>
<!ATTLIST pattern      name           CDATA          #REQUIRED>
<!ATTLIST pattern      abstraction    CDATA          #IMPLIED>
<!ATTLIST pattern      domain         CDATA          #IMPLIED>


<!-- ========== Context Element ============================ -->


<!-- A context represents the environment within which a
     pattern describes itself and is a general motivation for
     its existence.

     summary          A title or summary of the description

     description      A description of the context
-->
<!ELEMENT context (summary?, description)>


<!-- ========== Force Element ============================== -->


<!-- A forces element is a section that contains one or more
     forces.

       force            A force
-->
<!ELEMENT forces (force+)>


<!-- A force represents a motivation of a pattern. It
     essentially amplifies the problem a pattern is trying to
     address and then serves as a constraint on the solution.

     summary          A title of the force or a summary of the
                      description

     description      A description of the force
-->
<!ELEMENT force (summary?, description)>


<!-- ========== Problem Element ============================ -->
```

```
<!-- A problem represents a design need that is to be addressed
     by a pattern. It essentially distinguishes the use of one
     pattern over another.

     summary          Quick overview of the problem

     description      More detailed explanation of the problem
-->
<!ELEMENT problem (summary?, description)>


<!-- ========== Solution Element ========================== -->


<!-- A solution solves the problem described in a pattern. It
     is composed of a number of participants and defines the
     static structure and dynamic interactions of them

     summary          Quick overview of the solution

     description      More detailed explanation of the
                      solution

     participants     Participants or roles in the solution

     structure        Static structure of the solution

     collaboration    Dynamic interactions found in the
                      solution
-->
<!ELEMENT solution (summary?, description, participants,
                    structure, collaboration)>


<!-- A structure represents the static interaction of
     participants (as in a UML class diagram) in a solution.

     description      Description of the collaboration,
                      including how the participants interact

     artifacts        External resources that further describe
                      the collaboration. This could be a UML
                      class diagram.
-->
<!ELEMENT structure (description, artifacts?)>


<!-- A collaboration represents the dynamic interaction of
     participants (as in a UML sequence or collaboration
     diagram) in a solution.

     description      Description of the collaboration,
                      including how the participants interact
```

```
     artifacts        External resources that further describe
                      the collaboration. This could be a UML
                      sequence diagram
-->
<!ELEMENT collaboration (description, artifacts?)>


<!-- A participant represents a distinct role played by a
     component in the pattern solution. Each participant
     describes its general characteristics but does not place
     any constraints on how it may be realized.

     name             Name of the participant, which must be
                      unique among the others.

     required         Determines whether or not this
                      participant is required to complete the
                      solution.

     description      Description of the participant and its
                      role in the solution
-->
<!ELEMENT participant (description)>
<!ATTLIST participant    name           CDATA        #REQUIRED>
<!ATTLIST participant    required       %Boolean;    "true">


<!-- ========== Consequence Element ====================== -->


<!-- A consequences element is a section that contains one or
     more consequences.

       consequence      A consequence
-->
<!ELEMENT consequences (consequence+)>


<!-- A consequence represents a pro or con of pattern usage. It
     describes how a pattern supports its objectives and the
     trade-offs in doing so.

     summary          A title of the consequence or a summary
                      of the description

     description      A description of the consequence
-->
<!ELEMENT consequence (summary?, description)>


<!-- ========== Relationship Element ====================== -->
```

```
<!-- A relationships element is a section that contains
     references to one or more patterns that are related to
     this one.

     relationship     A pattern related in some way to this
                      one
-->
<!ELEMENT relationships (relationship+)>


<!-- A relationship represents a relationship between two
     patterns. A pattern relationship is purely descriptive,
     but it does have an attribute that specifies what type of
     relationship it is. This element would be used to refer to
     a like pattern or to describe a pattern nesting.

     namespace        Namespace of the related pattern

     name             Name of the related pattern

     type             Defines the type of relationship

     summary          A short phrase that describes the related
                      pattern

     description      Description of how the two patterns are
                      related
-->
<!ELEMENT relationship (summary?, description)>
<!ATTLIST relationship     namespace    CDATA      #REQUIRED>
<!ATTLIST relationship     name         CDATA      #REQUIRED>
<!ATTLIST relationship     type         %RelType   “reference”>
```

Strategy DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2001-2002 ObjectVenture Inc.  All rights
     reserved.

     This product or document is protected by copyright and
     distributed under licenses restricting its use, copying,
     and distribution. No part of this product or documentation
     may be reproduced in any form by any means without prior
     written authorization of ObjectVenture and its licensors,
     if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining a pattern strategy, which is one
     of several possible realizations or implementations of a
     pattern.

     To support validation of your pattern strategy file,
     include the following DOCTYPE element at the beginning
     (after the "xml" declaration):

     <!DOCTYPE strategy PUBLIC
       "-//ObjectVenture//DTD Strategy 1.0//EN"
       "http://www.objectventure.com/dtds/strategy-1_0.dtd">

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- This entity is a reference to an external DTD. It defines
     a number of common entity and element definitions that are
```

```
     used here and in the other pattern DTDs.
-->
<!ENTITY % common SYSTEM "common.dtd">
%common;


<!-- ========== Strategy Element ========================== -->


<!-- A strategy represents one of many possible implementations
     of a pattern solution, a building block for other
     strategies or an idiom. It serves as a bridge from the
     more abstract notion of a pattern to the more rigid world
     of components. A strategy can describe the design of a
     single component or a large framework of components. A
     strategy is role based, and each role defines restrictions
     on any component or element that may fill it. It is this
     role-based mechanism that gives strategies their greatest
     value; reuse of a design (which the strategy codifies) is
     gained by plugging in different components and elements in
     each role.

     namespace          A space within which the strategy name
                        must be unique

     name               Name of the strategy

     description        Description of the strategy

     authors            Authors of the strategy

     version            Version information for the strategy

     akas               Other names for the pattern

     keywords           Categorizations or classifications of
                        the strategy

     roles              Roles that define the strategy

     pattern-ref        The pattern that this strategy provides
                        an implementation for

     strategies         Other strategies that this one is
                        composed of

     artifacts          External resources that further
                        describe the strategy (i.e. UML
                        diagrams, graphics, etc.)
-->
<!ELEMENT strategy (description?, authors?, version, akas?,
                    keywords?, roles, pattern-ref?,
                    strategies?, artifacts?)>
<!ATTLIST strategy      namespace     CDATA       #REQUIRED>
```

```
<!ATTLIST strategy       name           CDATA        #REQUIRED>

<!-- ========== Roles Element ============================== -->

<!-- A roles element is a section that contains one or more
     component and connector roles that describe the
     collaboration of components within a strategy.

     component-roles  Component roles that the strategy is
                      composed of

     connector-roles  Connector roles that strategy is composed
                      of
-->
<!ELEMENT roles (component-roles, connector-roles?)>

<!-- ========== Strategies Element ======================== -->

<!-- A strategies element is a section that contains one or
     more strategy references.

     strategy-ref     A strategy reference
-->
<!ELEMENT strategies (strategy-ref+)>

<!-- A strategy-ref represents a reference to a pattern
     strategy.

     namespace        Namespace of the strategy

     name             Name of the strategy
-->
<!ELEMENT strategy-ref EMPTY>
<!ATTLIST strategy-ref    namespace      CDATA      #REQUIRED>
<!ATTLIST strategy-ref    name           CDATA      #REQUIRED>

<!-- ========== Parent Pattern Element ==================== -->

<!-- A pattern-ref represents the pattern for which this
     strategy provides a solution implementation. A strategy
     may realize only one pattern.

     namespace        Namespace of the pattern

     name             Name of the pattern
-->
<!ELEMENT pattern-ref (role-map*)>
```

```
<!ATTLIST pattern-ref    namespace      CDATA        #REQUIRED>
<!ATTLIST pattern-ref    name           CDATA        #REQUIRED>


<!-- A role-ref represents the mapping of this component role
     to a pattern participant. This element must not be used
     unless the strategy has a parent pattern.

     role-name        Name of the component role

     participant      Name of the participant or pattern role
                      that this component role fills
-->
<!ELEMENT role-ref EMPTY>
<!ATTLIST role-ref  role-name         CDATA          #REQUIRED>
<!ATTLIST role-ref  participant       CDATA          #REQUIRED>
```

<!-- ========== Component Role Element ========== -->

```
<!-- A component role represents a plug-in point in a strategy
     for a component. The role specifies an interface, so to
     speak, that a component must satisfy to fill the role. Any
     number of components may be swapped in and out of each
     component role, as long as they adhere to the specified
     interface.

     name             Name of the component role

     stereotype       Defines the type of component the
                      component role may be mapped to

     is-interface     Restricts the mapping of this component
                      role to component interfaces only

     multiplicity     Allows the role to be filled by more than
                      one component. This is useful in patterns
                      like Abstract Factory, where the concrete
                      factory role will be mapped to multiple
                      components.

                      The following values are available:
                        1     - One
                        #     - Any whole number > 1
                        *     - Many or more than one

     required         Determines whether or not this component
                      role is required to be filled when a
                      strategy is mapped

     description      Description of the component role

     extends          Inheritance relationships with other
```

```
                        component roles. All related roles must
                        have the same value for is-interface

                        If the component that fills a component
                        role is composed of a single class, then
                        it is required to subclass the component
                        that fills the parent component role.

     implements         Interfaces that this component role
                        implements. The associated component
                        roles must have is-interface set to true.

     attribute-roles    Child attribute roles

     operation-roles    Child operation roles

     tag-roles          Child tag roles
-->
<!ELEMENT component-role (description?, extends?, implements?,
                          attribute-roles?, operation-role?,
                          tag-roles?, participant-ref?)>
<!ATTLIST component-role  name           CDATA       #REQUIRED>
<!ATTLIST component-role  stereotype     CDATA       #IMPLIED>
<!ATTLIST component-role  is-interface   %Boolean;   "false">
<!ATTLIST component-role  multiplicity   CDATA       #REQUIRED>
<!ATTLIST component-role  required       %Boolean;   "true">


<!-- An extends element is a section that contains one or
     more components that a component must descend from
     to satisfy the component role. The associated component
     role must have is-interface set to false.

     component-role-ref  A parent component
-->
<!ELEMENT extends (component-role-ref +)>


<!-- An implements element is a section that contains one or
     more component interfaces that a component must implement
     to satisfy the component role. The associated component
     role must have is-interface set to true.

     component-role-ref  A component interface
-->
<!ELEMENT implements (component-role-ref +)>


<!-- An component-role-ref element represents a reference to a
     component role.

     namespace          Namespace of the component role, which is
                        not required if the component interface
                        is owned by the same strategy
```

```
     name             Name of the component role
-->
<!ELEMENT component-role-ref EMPTY>
<!ATTLIST component-role-ref   namespace   CDATA     #IMPLIED>
<!ATTLIST component-role-ref   name        CDATA     #REQUIRED>


<!-- ========== Attribute Role Element ==================== -->


<!-- An attribute-roles element is a section that contains one
     or more attribute roles.

     attribute-role   An attribute role
-->
<!ELEMENT attribute-roles (attribute-role+)>


<!-- An attribute role represents an attribute of a component.
     Each attribute role that is defined further restricts the
     components that a component role may be mapped to. Each
     required attribute role must be mapped to a valid
     attribute before the strategy is properly implemented.

     name             Name of the attribute role

     type             Type that an attribute must be to satisfy
                      the attribute role. Examples of Java
                      types include "java.lang.String" and
                      "boolean." The type may be represented
                      using an SCML substitution.

     stereotype       Defines the type of attribute the
                      attribute role may be mapped to

     visibility       Visibility that an attribute must have to
                      satisfy the attribute role. For example,
                      if the attribute role specifies a
                      visibility of "public", then it may only
                      be mapped to a public attribute.

     static           Determines whether or not the attribute
                      that the attribute role is mapped to must
                      belong to a component or an instance. For
                      example, an attribute role with static
                      set to "true" may not be mapped to an
                      attribute that is owned by an instance.

     constant         Determines whether or not the attribute
                      that the attribute role is mapped to must
                      be constant. For example, an attribute
                      role with constant set to "true" may not
                      be mapped to a mutable attribute.
```

```
     multiplicity     Allows the role to be filled by more than
                      one attribute. This is useful in patterns
                      like Value Object, where the role
                      representing data will be mapped to
                      multiple attributes.

                      The following values are available:
                        1     - One
                        #     - Any whole number > 1
                        *     - Many or more than one

     required         Determines whether or not this attribute
                      role is required to be mapped when its
                      parent component role is mapped to a
                      component

     description      Description of the attribute role
-->
<!ELEMENT attribute-role (description?)>
<!ATTLIST attribute-role  name           CDATA       #REQUIRED>
<!ATTLIST attribute-role  type           CDATA       #IMPLIED>
<!ATTLIST attribute-role  stereotype     CDATA       #IMPLIED>
<!ATTLIST attribute-role  visibility     %Access;    "public">
<!ATTLIST attribute-role  static         %Boolean;   "false">
<!ATTLIST attribute-role  constant       %Boolean;   "false">
<!ATTLIST attribute-role  multiplicity   CDATA       #REQUIRED>
<!ATTLIST attribute-role  required       %Boolean;   "true">


<!-- ========== Operation Role Element ==================== -->


<!-- An operation-roles element is a section that contains one
     or more operation roles.

     operation-role   An operation role
-->
<!ELEMENT operation-roles (operation-role+)>


<!-- An operation role represents a method of a component. Each
     operation role that is defined further restricts the
     components that a component role may be mapped to. Each
     required operation role must be mapped to a valid method
     before the strategy is properly implemented.

     name             Name of the operation role

     stereotype       Defines the type of method the operation
                      role may be mapped to

     visibility       Visibility that a method must have to
                      satisfy the operation role. For example,
```

```
                    if the operation role specifies a
                    visibility of "public", then it may only
                    be mapped to a public method.

     static         Determines whether or not the method that
                    the operation role is mapped to must
                    belong to a component or an instance. For
                    example, an operation role with static
                    set to "true" may not be mapped to a
                    method that is owned by an instance.

     return-type    Return type of that a method must have to
                    satisfy the operation role. For example,
                    an operation role with a return type set
                    to "boolean" may not be mapped to a
                    method with a return type of "int" or one
                    that has no return type. The return-type
                    may be represented using an SCML
                    substitution.

     multiplicity   Allows the role to be filled by more than
                    one attribute. This is useful in patterns
                    like Value Object, where the role
                    representing data will be mapped to
                    multiple attributes.

                    The following values are available:
                      1     - One
                      #     - Any whole number > 1
                      *     - Many or more than one

     required       Determines whether or not this operation
                    role is required to be mapped when its
                    parent component role is mapped to a
                    component

     description    Description of the operation role

     parameter-roles  Arguments that define part of the
                    operation role's signature

     body           Body of the operation role
-->
<!ELEMENT operation-role (description?, parameter-roles?,
                          body?)>
<!ATTLIST operation-role  name          CDATA       #REQUIRED>
<!ATTLIST operation-role  stereotype    CDATA       #IMPLIED>
<!ATTLIST operation-role  visibility    %Access;    "public">
<!ATTLIST operation-role  static        %Boolean;   "false">
<!ATTLIST operation-role  return-type   CDATA       #IMPLIED>
<!ATTLIST attribute-role  multiplicity  CDATA       #REQUIRED>
<!ATTLIST operation-role  required      %Boolean;   "true">
```

```
<!-- A body is the content of an operation role's body. It may
     contain source code, Source Code Macro Language (SCML) or
     a combination of both. Refer to the SCML specification for
     details on its use.
-->
<!ELEMENT body (#PCDATA)>


<!-- ========== Parameter Role Element ==================== -->


<!-- An parameter-roles element is a section that contains one
     or more parameter roles. The list of parameter roles may
     be ordered or unordered.

     parameter-role   A parameter-role

     ordered          Determines whether or not the list of
                      parameter roles must be mapped in order
                      to a method's arguments
-->
<!ELEMENT parameter-roles (parameter-role+)>
<!ATTLIST parameter-role  ordered       %Boolean;   "true">


<!-- A parameter role represents a parameter of a method. Each
     parameter role that is defined further restricts the
     methods that the operation role may be mapped to. Each
     parameter role must be mapped to a valid parameter before
     the strategy is properly implemented.

     name           Name of the parameter role

     type           Type of the parameter role. Examples of
                    Java types include "java.lang.String" and
                    "boolean." The type may be represented
                    using an SCML substitution.

     constant       Determines whether or not the paramter
                    that the paramter role is mapped to must
                    be constant. For example, a paramter
                    role with constant set to "true" may not be
                    mapped to a mutable paramter.

     description    Description of the parameter role
-->
<!ELEMENT parameter-role (description?)>
<!ATTLIST parameter-role    name        CDATA       #REQUIRED>
<!ATTLIST parameter-role    type        CDATA       #REQUIRED>
<!ATTLIST parameter-role    constant    %Boolean;   "false">


<!-- ========== Tag Role Element ========================== -->
```

```
<!-- A tag-roles element is a section that contains one or more
     tag roles.

     tag-role   A tag role
-->
<!ELEMENT tag-roles (tag-role+)>


<!-- A tag role represents a tag in a markup component (e.g.
     HTML, JSP). Each tag role that is defined further
     restricts the components that the component role may be
     mapped to. Each tag role must be mapped to a valid tag
     before the strategy is properly implemented.

     name             Name of the tag role

     stereotype       Defines the type of tag the tag role may
                      be mapped to

     prefix           Default tag library prefix (for JSPs) or
                      a namespace

     tag-name         Literal name of the tag

     multiplicity     Allows the role to be filled by more than
                      one tag.

                      The following values are available:
                        1     - One
                        #     - Any whole number > 1
                        *     - Many or more than one

     required         Determines whether or not this tag role
                      is required to be mapped when its parent
                      component role is mapped to a component

     description      Description of the tag role

     tag-attribute-role  An attribute of the tag role

     tag-role         A nested tag role
-->
<!ELEMENT tag-role (description?, tag-attribute*, tag-role*)>
<!ATTLIST tag-role      name           CDATA         #REQUIRED>
<!ATTLIST tag-role      stereotype     CDATA         #IMPLIED>
<!ATTLIST tag-role      prefix         CDATA         #IMPLIED>
<!ATTLIST tag-role      tag-name       CDATA         #IMPLIED>
<!ATTLIST tag-role      multiplicity   CDATA         #REQUIRED>
<!ATTLIST tag-role      required       %Boolean;     "true">


<!-- ========== Tag Attribute Role Element ================ -->
```

```
<!-- An tag-attribute-roles element is a section that contains
     one or more tag attribute roles. The list of tag attribute
     roles may be ordered or unordered.

     tag-attribute-role   A tag-attribute-role

     ordered              Determines whether or not the list of
                          tag attribute roles must be mapped in
                          order to a tag's arguments
-->
<!ELEMENT tag-attribute-roles (tag-attribute-role+)>
<!ATTLIST tag-attribute-role    ordered  %Boolean;   "true">


<!— A tag attribute role represents a markup tag attribute.
    Each tag attribute role that is defined further restricts
    the markup tags that the tag role may be mapped to. Each
    tag attribute role must be mapped to a valid tag attribute
    before the strategy is properly implemented.

     name           Name of the tag attribute role

     value          Value of the tag attribute role

     constant       Determines whether or not the tag attribute
                    that the tag attribute role is mapped to
                    must be constant. For example, a tag
                    attribute role with constant set to "true"
                    may not be mapped to a mutable tag
                    attribute.

     description    Description of the tag attribute role
-->
<!ELEMENT tag-attribute-role (description?)>
<!ATTLIST tag-attribute-role    name     CDATA       #REQUIRED>
<!ATTLIST tag-attribute-role    value    CDATA       #IMPLIED>
<!ATTLIST tag-attribute-role    constant %Boolean;   "false">


<!-- ========== Connector Role Element ==================== -->


<!-- A connector role represents a binary relationship between
     components. Each connector role has two end roles that
     must both be attached to a component role. Each connector
     role must be mapped to a valid relationship before the
     strategy is properly implemented.

     name             Name of the connector role

     required         Determines whether or not this
                      connector role is required to be filled
                      when the strategy is mapped
```

```
     description      Description of the connector role

     connector-end-role  One end of the connector role
-->
<!ELEMENT connector-role (description?, connector-end-role*)>
<!ATTLIST connector-role    name        CDATA        #REQUIRED>
<!ATTLIST connector-role    required    %Boolean;    "true">


<!-- A connector end role represents one end of a binary
     relationship between components. Each connector end role
     further restricts which relationship a connector role may
     be mapped to.

     name             Name of the connector end role

     multiplicity     Defines the required number of component
                      roles for this end of the connector. The
                      following multiplicities are allowed:
                        1    - One
                        #    - Whole number
                        0..1 - Zero or One
                        0..* - Zero to Many
                        1..* - One to Many
                        #..* - Whole number to Many
                        *    - Many

     navigable        Determines whether or not this end is
                      visible to the other

     aggregation      Defines the nature of this end role's
                      association with the other one

     changeability    Defines the mutability of the end role
                      (not of the component role that fills
                      it)

     visibility       Defines the access other roles have to
                      this connector end

     component-role-ref  Name of the target component role

     description      Description of the connector end role
-->
<!ELEMENT connector-end-role (description?)>
<!ATTLIST connector-end-role  name          CDATA    #REQUIRED>
<!ATTLIST connector-end-role  multiplicity  CDATA    #REQUIRED>
<!ATTLIST connector-end-role  navigable     %Boolean;   "true">
<!ATTLIST connector-end-role  aggregation   %Aggregation;
                                            "aggregation">
<!ATTLIST connector-end-role  changeability %Mutability;
                                            "read-write">
<!ATTLIST connector-end-role  visibility    %Access;  "public">
```

```
<!ATTLIST connector-end-role  target        CDATA    #IMPLIED>


<!-- A component-role-ref element is a reference to a component
     role.

     namespace        Namespace of the component role, which is
                      not required if the component role is
                      owned by the same strategy

     name             Name of the component role
-->
<!ELEMENT component-role-ref EMPTY>
<!ATTLIST component-role-ref  namespace    CDATA      #IMPLIED>
<!ATTLIST component-role-ref  name         CDATA     #REQUIRED>
```

Catalog DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2001-2002 ObjectVenture Inc.  All rights
     reserved.

     This product or document is protected by copyright and
     distributed under licenses restricting its use, copying,
     and distribution. No part of this product or documentation
     may be reproduced in any form by any means without prior
     written authorization of ObjectVenture and its licensors,
     if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining a pattern catalog.

     To support validation of your pattern catalog file,
     include the following DOCTYPE element at the beginning
     (after the "xml" declaration):

     <!DOCTYPE catalog PUBLIC
       "-//ObjectVenture//DTD Catalog 1.0//EN"
       "http://www.objectventure.com/dtds/catalog-1_0.dtd">

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- This entity is a reference to an external DTD. It defines
     a number of common entity and element definitions that are
     used here and in the other pattern DTDs.
-->
```

```
<!ENTITY % common SYSTEM "common.dtd">
%common;

<!-- ========== Catalog Descriptor Element ================ -->

<!-- A catalog descriptor holds the root catalog, sets its
     namespace and provides information about it.

     namespace      A space within which the root catalog name
                    must be unique

     authors        Authors of the catalog

     version        Version information for the catalog

     catalog        Root catalog

     artifacts      External resources that further describe
                    the catalog and its contents (i.e. UML
                    diagrams, graphics, etc.)
-->
<!ELEMENT catalog-descriptor (description?, authors?, version,
                              catalog, artifacts?)>
<!ATTLIST catalog-descriptor  namespace    CDATA     #REQUIRED>

<!-- ========== Catalog Element ============================ -->

<!-- A catalog groups a number of related patterns and
     strategies according to some criteria. There is no
     restriction on how they are grouped, so it could be by
     domain, company, abstraction level, etc. A catalog serves
     as the basis for packaging and exchanging patterns and
     strategies.

     name           Name of the catalog

     description    Description of the catalog

     catalogs       Nested catalogs

     patterns       Patterns that are included within the
                    catalog

     strategies     Strategies that are included within the
                    catalog
-->
<!ELEMENT catalog (description?, catalogs?, patterns?,
                   strategies?)>
<!ATTLIST catalog        name           CDATA        #REQUIRED>
```

```
<!-- A catalogs element is a section that contains one or more
     nested catalogs.

     catalog        A reference to a pattern catalog.
-->
<!ELEMENT catalogs (catalog+)>


<!-- ========== Pattern Reference Element ================= -->


<!-- A patterns element is a section that contains a reference
     to one or more patterns.

     pattern-ref    A reference to a pattern
-->
<!ELEMENT patterns (pattern-ref+)>


<!-- A pattern-ref element is a reference to a pattern that is
     included as part of the catalog.

     namespace      Namespace of the pattern

     name           Name of the pattern

     description    Description of the referenced pattern
-->
<!ELEMENT pattern-ref (description?)>
<!ATTLIST pattern-ref    namespace      CDATA        #REQUIRED>
<!ATTLIST pattern-ref    name           CDATA        #REQUIRED>


<!-- ========== Strategy Reference Element ================ -->


<!-- A strategies element is a section that contains a
     reference to one or more strategies.

     strategy-ref   A reference to a strategy
-->
<!ELEMENT strategies (strategy-ref+)>


<!-- A strategy-ref element is a reference to a strategy that
     is included as part of the catalog.

     namespace      Namespace of the strategy

     name           Name of the strategy

     description    Description of the referenced strategy
-->
```

```
<!ELEMENT strategy-ref (description?)>
<!ATTLIST strategy-ref   namespace      CDATA        #REQUIRED>
<!ATTLIST strategy-ref   name           CDATA        #REQUIRED>
```

Component DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2001-2002 ObjectVenture Inc.  All rights
     reserved.

     This product or document is protected by copyright and
     distributed under licenses restricting its use, copying,
     and distribution. No part of this product or documentation
     may be reproduced in any form by any means without prior
     written authorization of ObjectVenture and its licensors,
     if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining a component.

     To support validation of your component file, include the
     following DOCTYPE element at the beginning (after the
     "xml" declaration):

     <!DOCTYPE component PUBLIC
       "-//ObjectVenture//DTD Component 1.0//EN"
       "http://www.objectventure.com/dtds/component-1_0.dtd">

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- This entity is a reference to an external DTD. It defines
     a number of common entity and element definitions that are
     used here and in the other pattern/component DTDs.
-->
```

```
<!ENTITY % common SYSTEM "common.dtd">
%common;

<!-- ========== Component Element ========================== -->

<!-- A component represents an actual component. We do not
     describe a component's interface or internals here,
     because that is already done well through the particular
     component standard in use as well as UML itself. Instead,
     we describe authorship, versioning, external artifacts,
     etc.

     namespace        A space within which the component name
                      must be unique. This is usually a package
                      for Java components.

     name             Name of the component

     type             Component type. This will be based on
                      component stereotypes that are provided
                      in UML Profiles (see PCML specification
                      for more detail).

     description      Description of the component

     authors          Authors of the component

     version          Version information for the component

     keywords         Categorizations of the component

     artifacts        External resources that further describe
                      the component (i.e. UML diagrams,
                      graphics, etc.).
-->
<!ELEMENT component (description?, authors?, version,
                     keywords?, artifacts?)>
<!ATTLIST component          namespace      CDATA    #REQUIRED>
<!ATTLIST component          name           CDATA    #REQUIRED>
<!ATTLIST component          type           CDATA    #IMPLIED>
```

Strategy Instance DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2000-2002 ObjectVenture Inc.  All rights
     reserved.

     This product or document is protected by copyright and
     distributed under licenses restricting its use, copying,
     and distribution. No part of this product or documentation
     may be reproduced in any form by any means without prior
     written authorization of ObjectVenture and its licensors,
     if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining a component framework.

     To support validation of your implemented strategy file,
     include the following DOCTYPE element at the beginning
     (after the "xml" declaration):

     <!DOCTYPE istrategy PUBLIC
       "-//ObjectVenture//DTD IStrategy 1.0//EN"
       "http://www.objectventure.com/dtds/istrategy-1_0.dtd">

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- This entity is a reference to an external DTD. It defines
     a number of common entity and element definitions that are
     used here and in the other pattern/component DTDs.
-->
```

```
<!ENTITY % common SYSTEM "common.dtd">
%common;

<!-- ========== Strategy Instance Element ================ -->

<!-- An istrategy element is an instance of a strategy. It
     provides a particular mapping of components and their
     elements to all required roles of a strategy.

     namespace      Namespace of the strategy

     name           Name of the strategy

     description    Description of the strategy instance

     component-role-maps
                    Mappings of components to component roles

     connector-role-maps
                    Mappings of component relationships to
                    connector roles

     artifacts      External resources that further describe
                    the instance of the strategy (i.e. UML
                    diagrams, graphics, etc.)
-->
<!ELEMENT istrategy (description?, component-role-maps?,
                     connector-role-maps?, artifacts?)>
<!ATTLIST istrategy      namespace      CDATA        #REQUIRED>
<!ATTLIST istrategy      name           CDATA        #REQUIRED>

<!-- ========== Component Role Map Element ================ -->

<!-- A component-role-maps element is a section that contains
     one or more mappings of components to component roles.

     component-role-map  A mapping of components to a component
                         role
-->
<!ELEMENT component-role-maps (component-role-map+)>

<!-- A component-role-map represents the filling of a component
     role by one or more components.

     role-name      Name of the component role

     description    Description of the mapping

     mapped-components
```

```
                    Components that fill the role. A component
                    role may be filled by more than one
                    component if the component role's
                    multiplicity is greater than one.

     attribute-role-maps
                    Mappings of component attributes to
                    attribute roles

     operation-role-maps
                    Mappings of component methods to operation
                    roles

     tag-role-maps  Mappings of component markup tags to tag
                    roles
-->
<!ELEMENT component-role-map (description?, mapped-components?,
                              attribute-role-maps?,
                              operation-role-maps?,
                              tag-role-maps?)>
<!ATTLIST component-role-map  role-name    CDATA     #REQUIRED>


<!-- A mapped-components element is a section that contains one
     or more components that map to the same component role.

     mapped-component  A component mapped to a component role
-->
<!ELEMENT mapped-components (mapped-component+)>


<!-- A mapped-component represents a component that fills a
     component role.

     namespace      Namespace of the component. This would
                    likely be a package name for a Java
                    component.

     name           Name of the component
-->
<!ELEMENT mapped-component EMPTY>
<!ATTLIST mapped-component   namespace  CDATA        #REQUIRED>
<!ATTLIST mapped-component   name       CDATA        #REQUIRED>


<!-- ========== Attribute Role Map Element ================ -->


<!-- An attribute-role-maps element is a section that contains
     one or more mappings of component attributes to attribute
     roles.

     attribute-role-map  A component attribute filling an
                         attribute role
```

```
-->
<!ELEMENT attribute-role-maps (attribute-role-map+)>


<!-- An attribute-role-map represents the filling of an
     attribute role by one or more component attributes.

     role-name      Name of the attribute role

     description    Description of the mapping

     mapped-attributes
                    Attributes that fill the role. An attribute
                    role may be filled by more than one
                    component attribute if the attribute role's
                    multiplicity is greater than one.
-->
<!ELEMENT attribute-role-map (description?,
                              mapped-attributes?)>
<!ATTLIST attribute-role-map   role-name   CDATA     #REQUIRED>


<!-- A mapped-attributes element is a section that contains one
     or more component attributes that map to the same
     attribute role.

     mapped-attribute  A component attribute mapped to an
                       attribute role
-->
<!ELEMENT mapped-attributes (mapped-attribute+)>


<!-- A mapped-attribute represents a component attribute that
     fills an attribute role.

     name           Name of the component attribute
-->
<!ELEMENT mapped-attribute EMPTY>
<!ATTLIST mapped-attribute      name      CDATA      #REQUIRED>


<!-- ========== Operation Role Map Element ================ -->


<!-- An operation-role-maps element is a section that contains
     one or more mappings of component methods to operation
     roles.

     operation-role-map  A component method filling an
                         operation role
-->
<!ELEMENT operation-role-maps (operation-role-map+)>
```

```
<!-- An operation-role-map represents the filling of an
     operation role by one or more component methods.

     role-name      Name of the operation role

     description    Description of the mapping

     mapped-methods Methods that fill the role. An operation
                    role may be filled by more than one
                    component method if the operation role's
                    multiplicity is greater than one.
-->
<!ELEMENT operation-role-map (description?, mapped-methods?)>
<!ATTLIST operation-role-map  role-name     CDATA    #REQUIRED>


<!-- A mapped-methods element is a section that contains one or
     more component methods that map to the same operation
     role.

     mapped-method  A component method mapped to an operation
                    role
-->
<!ELEMENT mapped-methods (mapped-method+)>


<!-- A mapped-method represents a component method that fills
     an operation role.

     signature      Signature of the method

     parameter-role-map
                    A mapping of a method parameter to an
                    operation role parameter
-->
<!ELEMENT mapped-method (parameter-role-map*)>
<!ATTLIST mapped-method     signature     CDATA      #REQUIRED>


<!-- A parameter-role-map represents the filling of a parameter
     role by one or more component method parameters.

     role-name      Name of the parameter role

     param-name     Name of the component method parameter
-->
<!ELEMENT parameter-role-map EMPTY>
<!ATTLIST parameter-role-map  role-name     CDATA    #REQUIRED>
<!ATTLIST parameter-role-map  param-name    CDATA    #REQUIRED>


<!-- ========== Tag Role Map Element ====================== -->
```

```
<!-- A tag-role-maps element is a section that contains one or
     more mappings of component markup tags to tag roles.

     tag-role-map  A component markup tag filling a tag role
-->
<!ELEMENT tag-role-maps (tag-role-map+)>


<!-- A tag-role-map represents the filling of a tag role by one
     or more component markup tags.

     role-prefix    Tag role library prefix or namespace

     role-name      Name of the tag role

     description    Description of the mapping

     mapped-tags    Tags that fill the role. A tag role may be
                    filled by more than one component markup
                    tag if the tag role's multiplicity is
                    greater than one.
-->
<!ELEMENT tag-role-map (description?, mapped-tags?)>
<!ATTLIST tag-role-map     role-prefix     CDATA     #IMPLIED>
<!ATTLIST tag-role-map     role-name       CDATA     #REQUIRED>


<!-- A mapped-tags element is a section that contains one or
     more component markup tags that map to the same tag role.

     mapped-tag     A component markup tag mapped to a tag role
-->
<!ELEMENT mapped-tags (mapped-tag+)>


<!-- A mapped-tag represents a component markup tag that fills
     a tag role.

     prefix         Tag library prefix or namespace

     name           Name of the tag

     tag-attribute-role-map
                    A mapping of a component tag attribute to a
                    tag attribute role
-->
<!ELEMENT mapped-tag (tag-attribute-role-map*)>
<!ATTLIST mapped-tag        prefix       CDATA       #IMPLIED>
<!ATTLIST mapped-tag        name         CDATA       #REQUIRED>


<!-- A tag-attribute-role-map represents the filling of a tag
     attribute role by one or more component markup tag
     attributes.
```

```
     role-name      Name of the tag attribute role

     attribute-name Name of the component tag attribute
-->
<!ELEMENT tag-attribute-role-map EMPTY>
<!ATTLIST tag-attribute-role-map  role-name  CDATA   #REQUIRED>
<!ATTLIST tag-attribute-role-map
                            attribute-name  CDATA   #REQUIRED>


<!-- ========== Connector Role Map Element ================ -->


<!-- A connector-role-maps element is a section that contains
     one or more mappings of component connectors (or
     relationships) to connector roles.

     connector-role-map  A component connector filling a
                         connector role
-->
<!ELEMENT connector-role-maps (connector-role-map+)>


<!-- A connector-role-map represents the filling of a connector
     role by a component connector (or relationship).

     role-name      Name of the connector role

     description    Description of the mapping

     connector-end-role-map
                    A component filling a connector end role
-->
<!ELEMENT connector-role-map (description?,
                              connector-end-role-map+)>
<!ATTLIST connector-role-map   role-name   CDATA     #REQUIRED>


<!-- A connector-end-role-map represents the filling of a
     connector end role by a component. It essentially defines
     the target component and where the connector attaches
     itself to a the component. There must only be two of these
     per connector-role-map.

     role-name      Name of the connector end role

     description    Description of the mapping

     component-name Target of the connector end role

     component-namespace
                    Namespace of the component
```

```
     component-attribute-name
                    Where the connector end attaches itself to
                    the component
-->
<!ELEMENT connector-end-role-map (description?)>
<!ATTLIST connector-end-role-map
              role-name                    CDATA     #REQUIRED>
<!ATTLIST connector-end-role-map
              component-name               CDATA     #REQUIRED>
<!ATTLIST connector-end-role-map
              component-namespace          CDATA     #REQUIRED>
<!ATTLIST connector-end-role-map
              component-attribute-name     CDATA     #REQUIRED>
```

Palette DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- Copyright (c) 2001-2002 ObjectVenture Inc.  All rights
     reserved.

     This product or document is protected by copyright and
     distributed under licenses restricting its use, copying,
     and distribution. No part of this product or documentation
     may be reproduced in any form by any means without prior
     written authorization of ObjectVenture and its licensors,
     if any.

     THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR
     IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
     PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL
     OBJECTVENTURE INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
     INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
     GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
     BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
     LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
     OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
     POSSIBILITY OF SUCH DAMAGE.
-->

<!-- This is the DTD defining a component palette.

     To support validation of your component palette file,
     include the following DOCTYPE element at the beginning
     (after the "xml" declaration):

     <!DOCTYPE palette PUBLIC
       "-//ObjectVenture//DTD Palette 1.0//EN"
       "http://www.objectventure.com/dtds/palette-1_0.dtd">

     Version: 1.0
-->


<!-- ========== Common Types ============================== -->


<!-- This entity is a reference to an external DTD. It defines
     a number of common entity and element definitions that are
     used here and in the other pattern/component DTDs.
-->
```

```
<!ENTITY % common SYSTEM "common.dtd">
%common;

<!-- ========== Palette Descriptor Element ================ -->

<!-- A palette descriptor holds the root palette, sets its
     namespace and provides information about it.

     namespace      A space within which the root palette name
                    must be unique

     authors        Authors of the palette

     version        Version information for the palette

     palette        Root palette

     artifacts      External resources that further describe
                    the palette and its contents (i.e. UML
                    diagrams, graphics, etc.)
-->
<!ELEMENT palette-descriptor (description?, authors?, version,
                              palette, artifacts?)>
<!ATTLIST palette-descriptor  namespace    CDATA     #REQUIRED>

<!-- ========== Palette Element ============================ -->

<!-- A palette groups a number of related components according
     to some criteria. There is no restriction on how they are
     grouped, so it could be by domain, company, type,
     function, etc. A palette serves as the basis for packaging
     and exchanging a group of reusable components and
     frameworks. If instantiated strategies were included with
     the components or framework, then including catalogs
     containing the referenced patterns and strategies would
     not be uncommon.

     name           Name of the palette

     description    Description of the palette

     palettes       Nested palettes

     components     Components that are included within the
                    palette
-->
<!ELEMENT palette (description?, palettes?, components?)>
<!ATTLIST palette        name           CDATA        #REQUIRED>
```

```
<!-- A palettes element is a section that contains one or more
     nested palettes.

     palette        A reference to a component palette
-->
<!ELEMENT palettes (palette+)>


<!-- ========== Component Reference Element =============== -->


<!-- A components element is a section that contains a
     reference to one or more components.

     component-ref  A reference to a component
-->
<!ELEMENT components (component-ref+)>


<!-- A component-ref element is a reference to a component that
     is included as part of the palette.

     namespace      Namespace of the component

     name           Name of the component

     description    Description of the referenced component
-->
<!ELEMENT component-ref (description?)>
<!ATTLIST component-ref  namespace      CDATA        #REQUIRED>
<!ATTLIST component-ref  name           CDATA        #REQUIRED>
```