



Derivation of Domain Specific Design Patterns

Applications to design and
construction of SGS

Dan Port, USC Center for Software Engineering

GSAW-98 Presentation

Intelligence is the ability to perceive patterns. Genius is the ability to perceive patterns where the bulk of mankind cannot. Scholarship is the ability to perceive patterns where there aren't any.

-- Michael J. Moran

The Problem

✓ Full SGS frameworks

- too general
 - specialization needed
 - specialization of entire framework impractical
- costly to adopt in whole
- unclear how to adopt parts
- Ex. Netscape SuiteSpot, Lotus Domino, SSCS, Storm IMT-ISIS

✓ Object repositories/libraries

- too many parts
 - hard to find the “right” part
- parts too specialized or so general that heavy modification is required

What is a Design Pattern?

Design Patterns: A description of a solution to a common problem in terms of *components* and *objects*. (see [Gamma et. al. Sec. 1.1])

Examples:

- ✓ Gof4
 - Command, Reactor, Facade, Strategy
- ✓ Common OOD
 - Delegation, CB-Subtyping, CB-States, Metadata
- ✓ Architectural
 - MVC, MVCA, Representational Subsystems

Note that some of these patterns are context-sensitive and others are context-free

For a variety of reasons, developers rarely make direct application of context-sensitive design patterns and there aren't many context-free patterns.

Problems With General Design Patterns

Main reasons:

- ✓ No context given for which a patterns qualities (such as interface attributes, I/O parameters, etc.) can be coherently resolved.
- ✓ Scope of use is difficult to ascertain.
- ✓ Boundaries not always clear within system (does not always naturally decompose into patterns) - makes identification difficult and application artificial
- ✓ Generally too abstract

This often leads to attempts to “force fit” patterns into use, or worse, changing the requirements to accommodate them, resulting in wasted time and inelegant designs which lead to quality attribute consequences (e.g. “-ilities” such as maintainability, scaleability, extensionality, ...)

What is a DSDP?

Domain Specific Design Pattern: are collections of components and objects that represent, in software, well-defined (encapsulated, clear boundary, highly cohesive) partitions of a particular system domain.

Important principle: a system should be designed in a context larger than itself, but not so large that qualities can not be easily resolved.

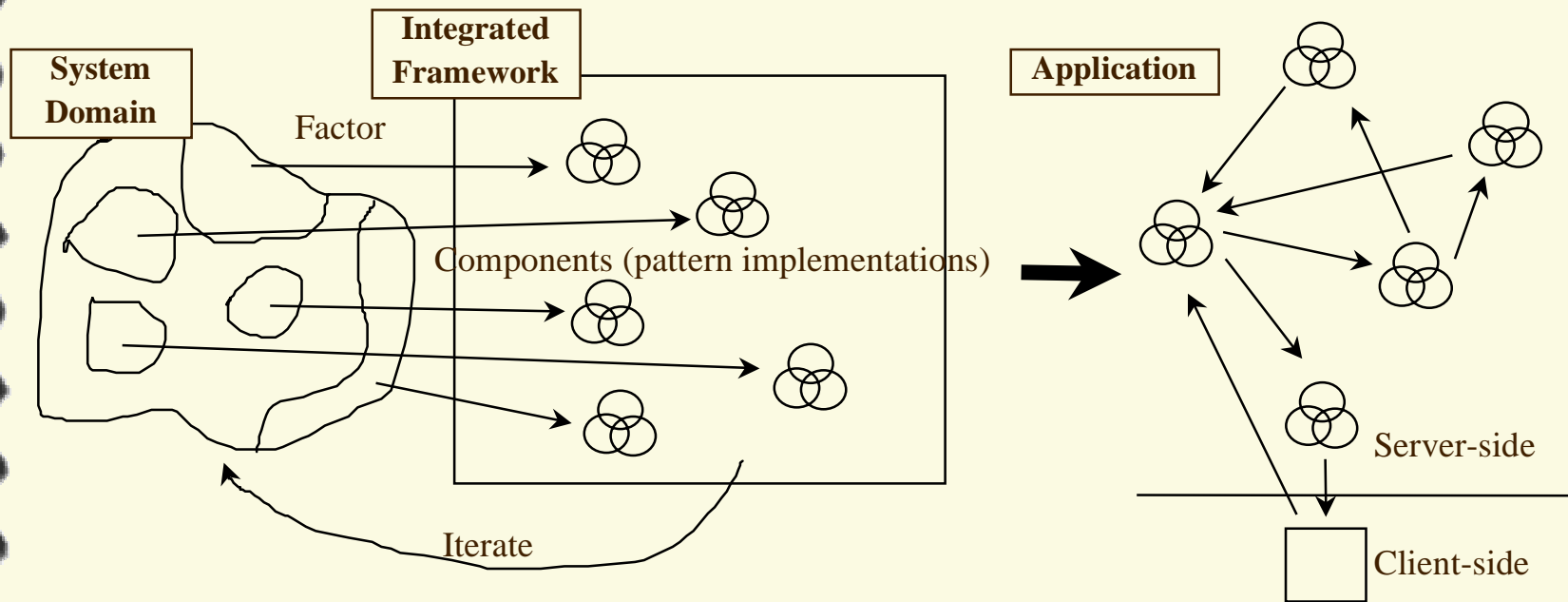
Goal: To clearly describe a flexible architecture in terms of well-defined, highly encapsulated, manageable parts that are in alignment with the natural constraints of the domain

Why Use DSDPs?

Main idea: discover the commonalties across the system domain, use them as natural partitions, create well-defined representations for them, then use them to construct systems within that domain.

- ✓ Assumption is that domain partitions are an inherent part of the domain and provide natural constraints for any application within that domain
- ✓ Partitions are small, well-defined, and manageable
- ✓ Qualities (such as relationships) are resolvable and meaningful within the domain context
- ✓ Partitions are easy to identify from sample applications
- ✓ Patterns can now be identified and fashioned to partitions

Domain Factoring: Decomposition followed by Classification



SGS Application Domain

Hardware Interfaces

RF Modulation/Demodulation
Ground-HW Control

Middle-ware

Command
Telemetry
Alarm Processing

High-level

Satellite Control
Payload Control
Ephemeris

General forms of DSDPs

name	uses a graphical user interface	how deployed - client or server	independent or dependent on other components to define it's operations	active in generating events and actions or passive and only responds to events and actions
Applications	GUI	client-side	independent	active
Mechanisms	no GUI	server-side	dependent	active
Glue	no GUI	server-side	dependent	passive
Components	N/A	N/A	dependent	N/A

Some SGS DSDP's

<u>Mechanisms</u>	<u>Glue</u>	<u>Components</u>
Ephemeris Prediction	Data	Command
Alarm Notification	Request	Messages
	Composition	
	Verification Method	

Value of DSDPs

Customer

- Identification of standard technology
- Evolution of systems based on unforeseen requirements
- Evolution of COTS products

Developer

- Manageable production line
- Optimize re-use
- Evolution/migration of components

Vendor

- Production focus
- Larger (more general) markets
- Not tied to particular customer
- Product line evolution