







🔷 Sun

To Know the Dependencies Is to Understand the Architecture

Neeraj Sangal

President Lattix, Inc. http://www.lattix.com

TS-6037

java.sun.com/javaone/sf

Dependencies Are Key to Managing Software Architecture

Prevent software architecture from eroding

Learn a new approach to representing and managing software architecture by utilizing inter-module dependencies. This session includes an actual demo and several real life examples.





Agenda

New Representation for Software Architecture Using Dependency Structure Matrix

Architectural Patterns

Design Rules

Architectural Evolution (Demo) Examples of Real Projects

Q&A



Agenda

New Representation for Software Architecture Using Dependency Structure Matrix

Architectural Patterns

Design Rules

Architectural Evolution (Demo) Examples of Real Projects

Q&A

The Dependency Model Approach

- Provides a precise big picture view of the architecture
- Enables explicit management of architectural evolution
- Lightweight approach that does not disrupt development
- Has been applied to dozens of applications written in the Java[™] language ranging in size from 100 classes to 20,000 classes





What Is a DSM?

		1	2	3	4
Module A	1			Х	Х
Module B	2		•	Х	
Module C	3	Х		•	Х
Module D	4				•

Fig 1: A Simple DSM

		1	2	3	4
Module D	1	-			
Module A	2	Х	-	Х	
Module C	3	Х	Х		
Module B	4			X	-

Fig 2: Block Triangular After Partitioning

		1	2	3
Module D	1			
Module A-C	2	Х	•	
Module B	3		Х	-

Fig 3: Lower Triangular

			1	2	3	4
Мо	dule D	1	-			
Þ	Module A	2	Х	-	Х	
Ó	Module C	3	Х	Х	-	
Мо	dule B	4			X	•

Fig 4: Hierarchical





What's a Dependency?

- Module A depends on a module B if there are explicit references in A to syntactic elements of B
- Simple but effective notion of dependency that works well for understanding design dependencies, in which modifications to one module might affect another
- Extraction of dependencies can be decoupled from their analysis



Java

Agenda

New Representation for Software Architecture using Dependency Structure Matrix

Architectural Patterns

Design Rules Architectural Evolution (Demo) Examples of Real Projects Q&A



Architectural Patterns—I

\$ro	ot		-	2	ω	4	G
	+ application	1					
con	+ model	2	37	12			
n.exa	🕂 domain	3	17	29	-		
com.example	+ framework	4	75	53	42	-	
æ	+ util	5	10	13	16	13	1.

Layered System

\$ro	ot		-	2	ω	4	თ
·	+ application	1	•				
con	+ model	2	37				
com.example	🕂 domain	3		29	-63		
amp	+ framework	4			42	a.	
æ	+ util	5		8.1		13	

Strictly Layered System



Architectural Patterns—II

6

7

8

9

1 1 1 1 1 1

1 1

\$root				243	N	c	с. С	4		л	
- + application		1			-	Ĩ	_			1	
S model		4	2	37						1	Imperfectly Layere
g 🗄 domain			3	17	26	5	22				System
example domain + framework		2	1	75	53	3 4	40				
e util		2	5	11	13	3 1	16	13	5		
		_									Change Propagator
project		-	2	ω	4	σ	თ	7	8	۵	Dropostor
+ actions	1								-	10	Propagator
+ events	2	1			2			1		2	
DefaultWorkspaceManager	3						1			1	
DefaultWorkspaceContext	4									1	
Partitioner	5								1	1	

÷

1

1

1

1



ProjectLoader

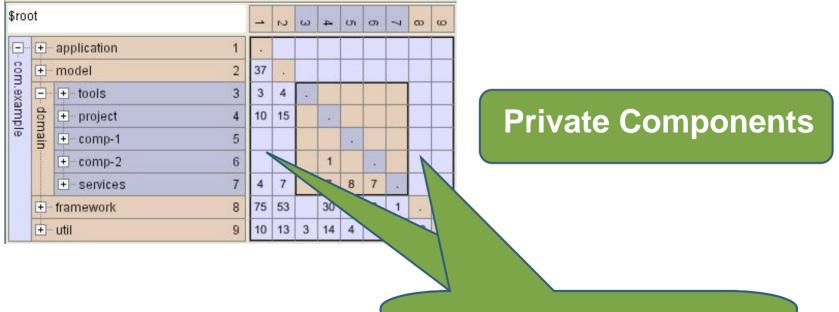
ProjectUpdater

ProjectView

Project



Architectural Patterns—III



Not Visible Outside "Domain"

Sun.

Java

Agenda

New Representation for Software Architecture using Dependency Structure Matrix

Architectural Patterns

Design Rules

Architectural Evolution (Demo) Examples of Real Projects Q&A





Design Rules

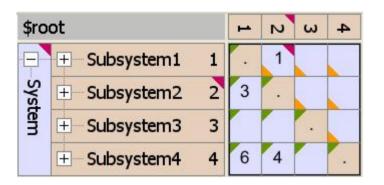
- Succinct specification of acceptable and unacceptable dependencies between subsystems
- Each cell of the DSM represents design intent
- DSM offers a powerful way to visualize and specify design rules

Dependency Model = DSM + Design Rules





Design Rules



DSM with Rules View

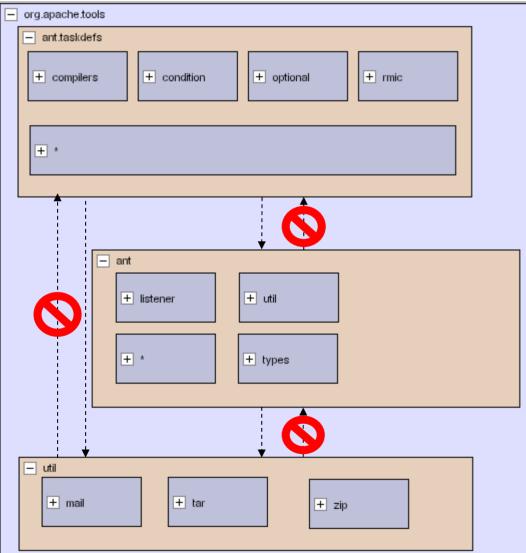
Green Triangle—Dependency Acceptable Yellow Triangle—Dependency Unacceptable Red Triangle—Rule Violation Discovered

\$ro	ot		-	Ν	ω	4	л
-	+ application	1	•	-			
OT	+ model	2	37	•			
com.example	+ domain	3	17	26			
	+ framework	4	75	53	40		
n	+ util	5	10	13	16	13	

Rules for Layering

- 1. \$root can-use \$root
- 2. model cannot-use application
- 3. domain cannot-use application, model
- 4. framework cannot use application, model, domain
- 5. util cannot-use application, model, domain, framework

ANT Conceptual Architecture



Layered **Architecture** with Three **Subsystems Tasks Use** Common Infrastructure Key Goal: Allow Independent **Development** of Tasks

lava

DEMO

The Evolution of ANT

2006 JavaOneSM Conference | Session XXXX | 16 java.sun.com/javaone/sf

Java

Agenda

New Representation for Software Architecture using Dependency Structure Matrix

Architectural Patterns

Design Rules

Architectural Evolution (Demo)

Examples of Real Projects

Q&A



Example: JUNIT

\$ro	ot	H	Ν	ω	4	ы	6
	🛨 awtui 1	•					
junit	🛨 swingui 2		•				
Ŧ	± textui 3			•			
	± extensions 4		1				
	± runner 5	3	13	5			
	+ framework 6	4	19	8	13	6	10

- junit		
+ awtui	+ swingui	+ textui
+ extensions		
+ runner		
+ framework		

A Layered System with Independent User Interfaces



Example: jEdit

org.	gjt.sp	н	2	ω	4	ы	6	7	ø	9	10	11	12	13	14	15
	🛨 print 1	·														
jedit	proto.jeditresource 2		•													
	+ help 3			•						1				1		
	+ options 4									2				1		
	🛨 menu 5					•								4		
	+ browser 6			1		7	•							3		
	± search 7						3							9		
	🛨 gui 8			2	23	5	7	12	•		6		2	42		1
	🛨 pluginmgr 9				2				1	•				1		
	± textarea 10					1		13	11		•	1		21		
	± buffer 11				1				1		4	•	1	13		
	±−io 12			4	1	4	29	9	3	2		3	X	16		
	±-* 13	11	4	17	103	59	41	51	138	24	31	19	25	•	2	22
	± syntax 14	3			4				1		6	1		16	•	
	± msg 15			1		2	4	3	4	2			3	25		•

Monolithic System ?



رنگ Java

Example: NetBeans[™] Platform

\$rc	00	ot		4	2	ω	4	л	б	۲	ω	9	10	11	12	IJ	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	• • [•	+ enumeration	ns 1	•																										
Ę	Ŀ	+ compat	2																											
Ļ.	• • [•	+ autoupdate	3																											
Ę	Ŀ	+ javahelp	4																											
-	• • [•	+ favorites	5					•																						
Ë	Ŀ	+ multiview	6																											
5	Ŀ	+ ProcSup	7																											
-	• • [•	+ ··· ui	8								•																			
Core	Ŀ	+ windows	9																											
GUI	Ŀ	+ core	10		1	3				4		13				1														
		- • + • tabcont	rol 11									38																		
	=	🗄 🛨 🖓 plaf	12										1																	
Ļ.	• • [+ settings	13																											
Libraries	Ŀ	+ actions	14		4			3			1	2	1																	
aries	Ŀ	+ loaders	15			7	4	16				43	89			12														
	Ŀ	+ masterfs	16																											
	Ŀ	+ text	17						5								1	19												
		- · · · + · · · progres	; 18			12	2			3																				
	001	ନ୍ତ୍ର 🗄 window	s 19		3	4	1	5	18	5	2	91	27	3		1	9	17		10	1									
		+ Node S	stem 20		10	42	8	39		29	2	69	210			19	54	213		1		7								
		. + ⊂ dialogs	21		3	57		4		4		20	25				3	25		5	4		11							
		+ awt	22			18	2	3	7	1		12	26	8			16	12		5	1	2	20	2						
	Ŀ	+ options	23			1				1			1			1			1	2			1							
	Ŀ	+ queries	24					1											2											
-	• • [•	+ startup	25			4							33															•		
ŝ	E	+ File System	26		13	27	1	15		11	4	41	94			36		151	95								7	71		
:	E	+ Module Syst	em 27		23	185	37	28	27	77	7	155	375	7		54	134	278	56	47	10	44	271	15	35	8	14	157	67	

اللي Java



Conceptual Architecture for NetBeans™

- \$root	
+ IDE Lite + Optional + Deprecated	
Core GUI + ui + windows + core + Swing Components - Libraries + settings + settings + loaders + text + GUI + masterfs	Precise Big Picture View Derived From The Dependencies View Shows Accurate Layering And Vertical
+ queries + options	Splitting
 NB Module System + startup + File System + Module System 	





Summary: Big Picture View That Scales

- Highly scalable—Represent massive systems to give you a precise big picture view
- Formalize design intent and prevent architectural erosion
- Easy to adopt—Use it at any stage of the lifecycle
- Critical visibility of the architecture is achieved very quickly—try it out on your own software!





(jij) Java

For More Information

- Information on DSMs: http://www.dsmweb.org
- Neeraj Sangal, Ev Jordan, Vineet Sinha, Daniel Jackson, "Using Dependency Models to Manage Complex Software Architecture", OOPSLA 2005 http://sdg.lcs.mit.edu/pubs/2005/oopsla05-dsm.pdf
- Steven D. Eppinger, "Innovation at the Speed of Information", Harvard Business Review, January 2001
- Baldwin, C.Y. and Clark K.B., The Power of Modularity Volume 1, MIT Press, Cambridge, MA, 2000
- Lattix: http://www.lattix.com





Neeraj Sangal

2006 JavaOneSM Conference | Session XXXX | 24 java.sun.com/javaone/sf









🔷 Sun

To Know the Dependencies Is to Understand the Architecture

Neeraj Sangal

President Lattix, Inc. http://www.lattix.com

TS-6037

java.sun.com/javaone/sf