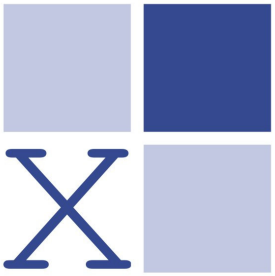
The Lattix™ Approach

DSM for Managing Software Architecture

Whitepaper

Nov 2004



Lattix DSM for Software Architectures

The Lattix LDM™ solution employs a unique and powerful strategy for communicating, visualizing, analyzing, and managing a software system's architecture. Lattix allows the architect to formalize and enforce the overall architecture of a system through design rules, and to leverage DSM as a key component of Lattix's approach for controlling the complexity of a large software system.

The Lattix approach allows developers of large software systems to formalize, understand, communicate and maintain their architecture. Further, this architecture is easy for managers to understand which makes it easier for business issues to be incorporated in the development process.

Since software development is a highly iterative process, the management of software architecture impacts the entire software life cycle. Lattix products help understand the impact of requirements, maintain the integrity of the architecture during the code implementation process, and simplify the test requirements. As a result, Lattix solutions have wide applicability.

Lattix Technology

Lattix believes that the management of dependencies between modules is critical to the management of software projects. This is consistent with cutting edge research now coming out of leading institutions such as MIT and Harvard. Lattix's patent pending approach is a new technology that allows DSM techniques to be adopted for the management of software development. DSM is a powerful and simple construct by which developers can communicate the architecture of complex products. It is also an artifact that enables managers to understand the impact of future revisions and enables them evaluate the risks associated with the development tasks. Lattix technology allows DSMs to be constructed automatically, and allows the architectural intent to be specified formally.

How Lattix defines Software Architecture

Lattix defines software architecture of a system as consisting of three primary elements:

- The decomposition of a system into its subsystems
- The visible properties of those subsystems
- The description of the dependencies between those sub-systems.

Decomposition of a system is a natural mental process that we use to manage complexity by subdividing that system into less complex subsystems. Further, when subsystems are chosen to represent clear abstractions, it becomes easier for programmers to comprehend and organize the information associated with those subsystems. Finally, keeping the dependencies between these subsystems well defined reduces the number of concerns that each programmer has to manage during development.

Benefits of Lattix DSM representation

The Lattix LDM solution is a unique and extremely powerful approach for maintaining the architecture of a software system. Some of the benefits that using DSM affords the Lattix solution include:

- Visibility into the architectural consistency of a software system
- Compact and precise representation
- Scalable system representation which enables visibility into very large systems
- Analysis algorithms to confirm designed architectural patterns, or discover previously unknown architectural patterns
- An ideal starting point for formalizing architecture through design rules

Using a DSM to Represent Architecture

Lattix's approach to managing the software architecture has evolved from research now coming out of leading institutions such as MIT, Harvard, and University of Illinois. MIT's Design Structure Matrix (DSM) group has a web site (http://www.dsmweb.org) and a group of practitioners who have applied dependency analysis to the study and streamlining of complex systems at many of the largest companies in the world. Professor Daniel Jackson's work at MIT has also focused on the importance of understanding dependencies.

Lattix has applied DSM techniques for the management of software development. DSM is a simple but powerful construct by which developers can communicate the architecture of complex products. It is also an artifact that enables managers to understand the impact of future revisions and enables them evaluate the risks associated with development tasks. Lattix's LDM application automatically extracts dependencies from a wide variety of software. Furthermore, the initial DSM is constructed to reflect the current hierarchy that is already in place. For instance in the case of Java, the initial DSM follows the package structure, for .NET the initial DSM reflects the name space structure, for C/C++ the initial name space reflects the file and directory structure etc.

Furthermore, Lattix allows the user to change the DSM hierarchy to more accurately reflect the architectural intent even when that intent is not explicit in the code. This is frequently an iterative process where architects gain new insights about their systems and refine the DSM to accurately represent their architecture. Lattix has further extended DSM to allow architects to establish design rules, which enable them to specify the nature of the dependencies.

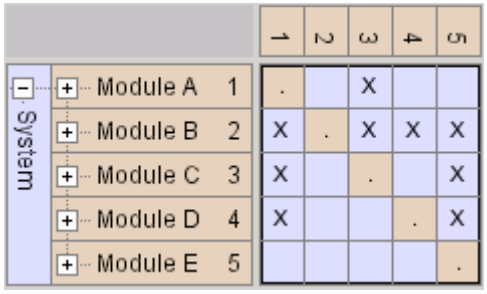


Figure 4: A Simple Lattix DSM

This figure shows that a system has been decomposed into five modules: Module A, Module B, Module C, Module D, and Module E. The dependencies for Module A are in column 1 which shows that Module A depends on Modules B, C, and D. This is shown by placing an X in rows 2, 3, and 4, which corresponds to Modules B, C, and D's rows. All of the dependencies indicated in the above DSM are as follows.

- Module A depends on Module B, C, and D
- Module B depends on No other modules
- Module C depends on Module A and B
- Module D depends on Module B
- Module E depends on Modules B, C, and D

This example is kept simple to illustrate the DSM concept. Actual DSMs will be larger. One of the major benefits of this approach is that it scales much better in terms of visualization and maintainability. An equivalent representation in terms of a directed graph (boxes for sub-systems and arrows between them to indicate dependency) quickly

becomes cluttered and difficult to understand. Because software structure (organization) and dependency information is captured automatically, the Lattix DSMs are easy to create and maintain. The Lattix solution does not force the architects or developers to maintain redundant information about the system, but rather makes that information available for them to leverage.

Using a DSM to Analyze an Architecture

A crucial benefit to DSM modeling is that you can use the matrix representation of a system to analyze the interrelations of that system, to see information flows, and to even discover previously unknown patterns in that system and its architecture. The use of Binary Matrices to aid in system analysis dates back to the 60's/70's with the work of John Warfield, and the DSM name and approach was pioneered by Don Steward in the 80's. However, the approach did not begin to gain momentum until the 90's.

Once represented in a DSM, a system's "natural" subsystems can be determined algorithmically. These algorithms are known as DSM Partitioning algorithms. Executed on a DSM, a partitioning algorithm will re-order the elements of the DSM such that organizational patterns and subsystems become apparent. The overall goal of DSM Partitioning algorithms is to order a system starting with the subsystems that use the most other subsystems and ending with the subsystems that provide the most to other subsystems, while grouping systems that have interdependence (cycles) together in the ordering.

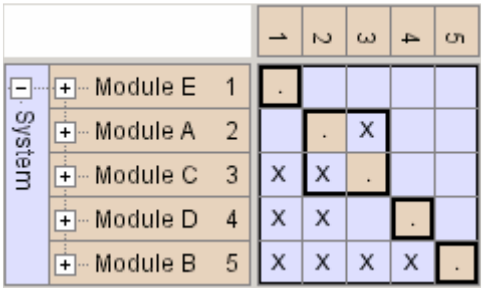


Figure 5: A Partitioned Simple Lattix DSM

Moreover, there are patterns that emerge and are visible in a DSM that can't be seen easily in a directed graph representation. DSM algorithms along with recognizing important patterns and using LDM to see important information about a system help the system's architect pursue initiatives when analyzing an existing system, or perhaps mining an existing system for reusable components or layers.

Some of the goals that an Architect might pursue include:

- **Minimize dependency cycles**, especially between large subsystems. This is seen in a DSM as striving for a Lower Triangular or Block Triangular form
- **Maintain layering**. Layers in a system are also visible from a Block Triangular representation. In the very simple example above, there are 4 layers, Module E is at the top, then Modules A and C form the next lower level, then Module D forms the 3rd layer, and Module B is the bottom-most level.
- **Component Independence.** Maintain the independence of components or limit the visibility of components in order to maintain modularity.
- **Manage use of external libraries**. Track usage of external libraries for appropriate "physical" layering, assure only approved license libraries are used

- **Discover redundant implementation.** Uncover potentially redundant implementations. By observing usage patterns of external facilities (like jdbc or odbc for database access), or similarly for well understood internal technologies, multiple implementations using these facilities can indicate redundant work.

Using a DSM to Manage Architecture

With the DSM representation providing a blueprint or a map of a software system's architecture, it is an ideal place to begin formalizing design rules that have previously been stored in seldom updated architectural specifications, or simply in the minds of the architects.

With the context of a map of the existing subsystems, their current decomposition, and the existing dependencies between subsystems, this framework is ready for codifying the architect's intent through a set of formal design rules. Typically, a design rule is applied to a subsystem and is then inherited by every subsystem contained within it. This allows a widely pervasive rule to be applied simply by clicking on a single cell of the DSM.

The Lattix LDM solution can then inform or enforce when developers inadvertently create architectural violations.

Formalizing Design Intent and applying Design Rules

The Lattix Solutions benefit development organizations by automatically generating a DSM blueprint of a software system, allowing visibility into a system's architecture that previously did not exist. Once software architects have greater visibility over their system, and greater ability to perform system architecture analysis, software architects can then begin making architecture changes, and putting mechanisms in place to preserve the integrity of their software architecture.

Lattix Solutions allow the architect to formalize their architectural intent, and to codify that architectural intent with rules such as ones that permit references or disallow references between subsystems. With LDM, the architect can establish, and enforce, rules that reflect intelligent abstraction, appropriate layering, and componentization. For example, if an architect wanted to enforce that all database access pass through a single module, they could easily do so with Lattix design rules. This has the obvious benefit that when the software system needs to add support for a new database, the access to databases is effectively encapsulated, and therefore the level of effort for the change is minimized to the work necessary to support the new database. This power extends to any type of software factoring that an architect would want to communicate or enforce.

The DSM representation lends itself very well to a comprehensible and easy to use mechanism to create and maintain design rules for a software system. Rules are typically applied to subsystems (modules), with all children subsystems inheriting the rules of their parent. Frequently, the critical rules are simply constraints which disallow dependencies between specific subsystems. A DSM representation makes clear, with precision, what rules are in place between any two modules, at any level of the system.

The figure below shows a simple DSM with 2 example rules shown in callouts. The rules in this example which disallow dependencies are enforcing the intended layering of subsystems.

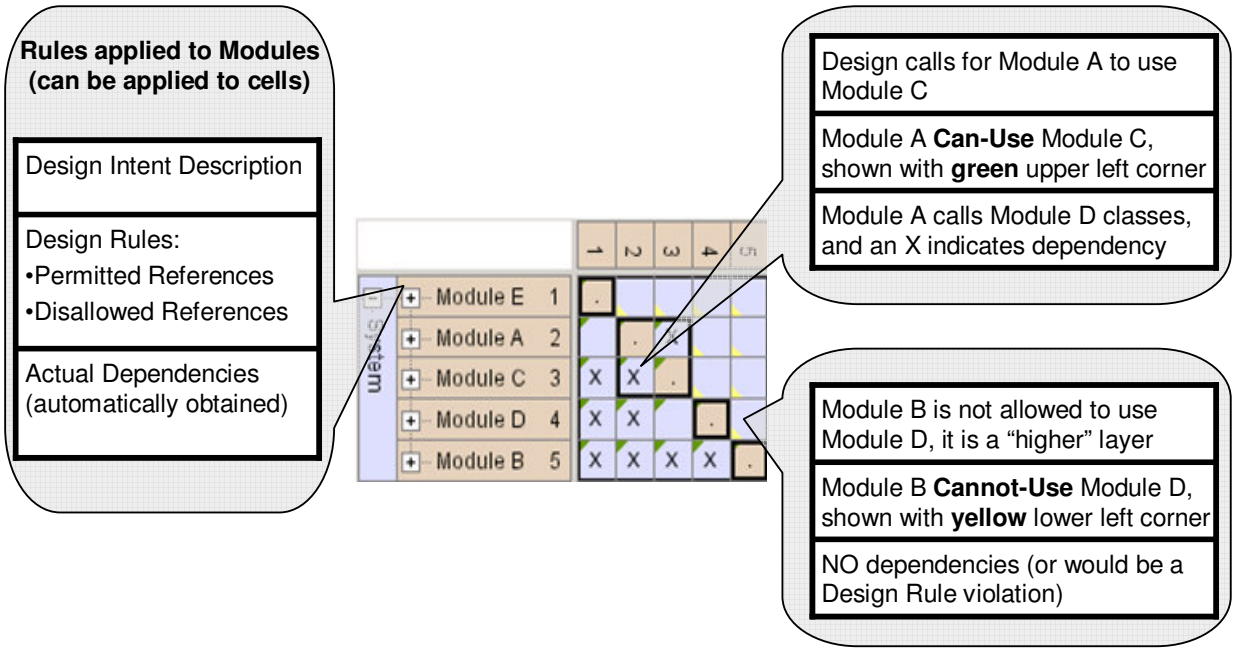


Figure 7: Formalizing architectural intent and applying Design Rules

Forward Engineering

As Architects conceive changes to their system, they can model those changes using LDM's forward engineering features. They can edit the structure of the software system stored in a LDM project to do things like splitting subsystems into multiple subsystems, aggregating two subsystems into one, or simply moving subsystems from one place to another. During these "what if" exercises which eventually lead to a "should be" architecture, the architects are also building the blueprint for the coding changes that will implement this new architecture.

DSM and UML

One of the major benefits of this representation is that it scales well. The critical design intent information that architects maintain in a DSM is currently not maintained formally anywhere. What this means is that DSMs do not suffer from a key difficulty that conventional methods such as UML models do – they do not duplicate the detailed implementation (the software implementation) and they remain synchronized with the code whereas UML models become unsynchronized in even the most rigorous development organizations. UML models also become much harder to comprehend as the scale increases. UML is a useful modeling language for capturing detailed design and use case information, and its strength is in the fine grained model of smaller portions of a large system. Accordingly, we see DSM and UML as complimentary modeling approaches, with DSM helping practitioners with the big picture of a system, and UML providing the detailed picture of facets of that system.

Terms

Lower Triangular Form

In this form, all dependency X's appear below the diagonal of the DSM. That means that all dependencies in the system "look up", and there are no feedback loops.

Block Triangular

This form is similar to the "Lower Triangular" form, except that dependencies do appear above the Identity Diagonal. However, those feedback loops appear as part of a distinct subsystem blocks, and the feedback is only between members of that subsystem. Note, if a block triangular DSM was constructed at the level of the subsystems, it would again be in a lower triangular form.

Block Diagonal

This form is similar to the "Block Triangular" form, except that there are empty cells above and BELOW the Diagonal outside of the interdependent blocks. When there are several subsystems that have no interdependencies, they are in essence modular components, and they will appear in Block Diagonal form.

Feedback Loops

Feedback loops are when, after DSM partitioning is complete, there are dependencies above the Identity Diagonal. This indicates that two subsystems are "coupled" in that they each have a dependency on the other. This type of dependency is often referred to as a "cycle".

Identity Diagonal

In a DSM, there is a cell that is the "dependency" a module has on itself. Generally, this is not considered a material dependency, and is usually either ignored, or assumed that a module depends upon itself. This diagonal is also where in a binary matrix, if all values are 1, then this is an identity matrix, which when multiplied by any other matrix will produce that other matrix

Example: Apache ANT tool (rev 1.5.1)

For a real example of a DSM for a software system's architecture, the open source built tool, ANT provides a reasonably typical example. Below in Figure 6 is a partitioned DSM for apache.org's ANT 1.5.1. This version of ant is made up of 20 packages, which contain more than 400 classes.

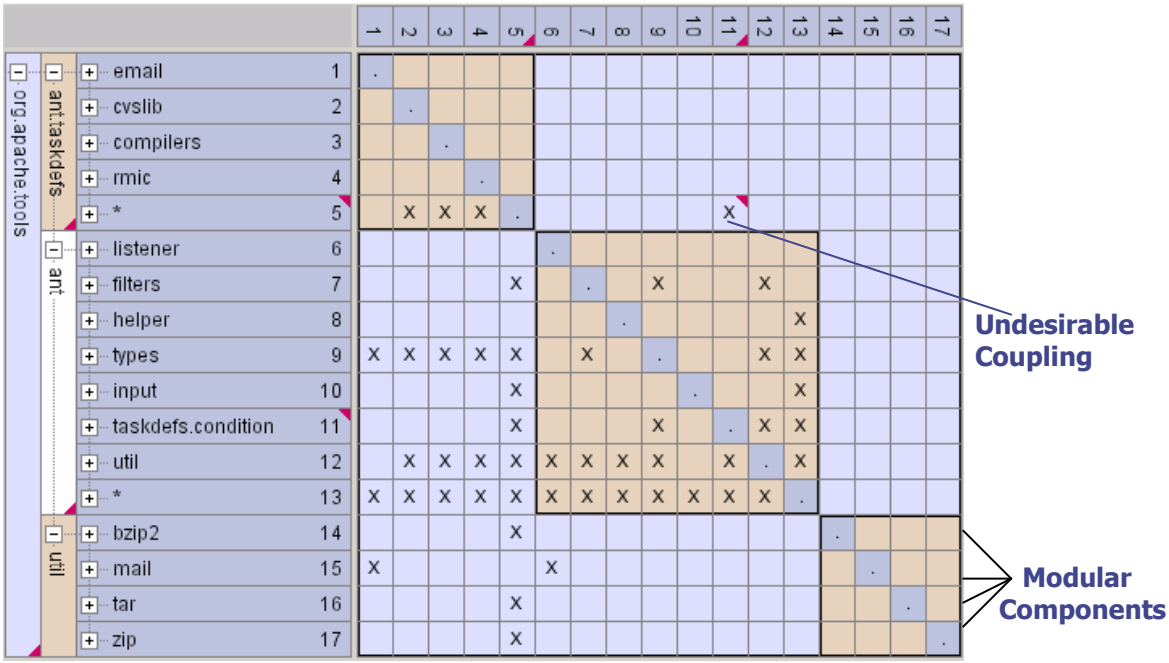


Figure 6: Partitioned DSM for Ant in Block Triangular form

Based on analysis of the DSM in Figure 6, it was possible to partition ANT into three subsystems, as depicted above. These subsystems are layered in order where the lower layers are independent of the higher layer. It can be immediately observed that this architecture is not completely consistent. Note the cell with red upper right corner indicating a rule violation. The X (row 5, column 11) shows a feedback loop existing outside of the existing subsystems.

Ant designers intended the layering to allow additional tasks be developed by multiple developers simultaneously by keeping the infrastructure independent of tasks. The DSM shows that they only partially succeeded in this objective. The conditional tasks are used by the engine and therefore need to be in the Engine. Additionally, the DSM highlights how intertwined the types subsystem is in the framework. This means that developers of newer types need to understand and deal with the entire ant infrastructure. If future improvements require parallel and easy development of new types then it would be advisable to define a component architecture for types.

More about DSM

More information can be found about DSM at MIT's website, http://www.dsmweb.org.

