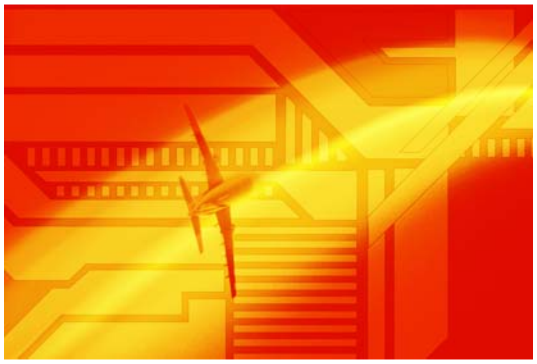
Model-driven systems development from IBM

Global markets are increasingly competitive; our products need to be more innovative to differentiate themselves in this environment. At the same time, systems are becoming increasingly complex. Increasing language capabilities, rapidly changing technology and the global flow of information are a few of the drivers of this complexity. Along with increasing complexity is an increase in the pace of change, which is creating a need to deliver more capability in less time. Further, systems are becoming more integrated and interdependent. Traditional systems development methods are not able to keep pace with these demands.

Within the aerospace and defense markets, the changes in systems development are especially dramatic because of the changing nature of threats coupled with rapid changes in technology.

Defense systems must become more agile and capabilities deployed more quickly to address today's complex threats. Thus, our development methods must help us integrate and deploy complex and scalable functionality more quickly.



Doing more for less

Software permeates everything, including aerospace and defense systems. Software development productivity has improved, but the software has ballooned in size. Systems today require constant updates, yet development organizations are expected to reduce the total cost of ownership of these systems. We must do more for less: innovate to stay ahead of the competition, but manage risk; meet new technical challenges, but manage cost.

The approach to system development and delivery is changing in fundamental ways. Historically, development life cycles were lengthy for the highly customized, proprietary solutions that were needed and demanded. Costs were a secondary concern.

Today, fewer components provide more functionality, yet their underlying code counts are significantly higher, adding to their complexity, development and management costs. In addition, systems must integrate with existing and future systems instantly. To succeed in today's rapidly changing and increasingly complex world, we must find ways to manage this complexity to deliver innovative systems to market more rapidly and at lower costs.

What makes systems complex

Systems can be characterized by their attributes, which fall into two groups:

- ''Black box'' attributes—externally observable characteristics, including the services the system provides.
- ''White box'' attributes—resources that make up the system. The system's white-box resources are encapsulated in its black-box services.

Systems development may be thought of as the specification of the black-box attributes of the system (that is, its requirements) and the delivery of the integrated system components that meet those requirements. A complete MDSD approach thus combines strong requirements management and modeling solutions to ensure that the delivery of the system is in line with expectations. With MDSD, we first break open the black box, and look at the system as a white box—this represents a transformation. Next, we break down (or, "decompose") the system into pieces to understand how the pieces work together to meet the black-box requirements, thereby deriving requirements on the pieces.

In building systems, we are faced with two different kinds of complexity:

- Creative/dynamic
- Transactional

Creative/dynamic complexity arises because teams of people need to work together creatively to architect optimal, robust systems. Transactional complexity arises when we try to manage all the components that make up a complex system.

Creative/dynamic complexity can be managed with a governance process. Model-driven systems development (MDSD) enables us to manage transactional complexity.

Requirements management—integral to successful systems development

Requirements represent the client (external and internal) through all phases of the development life cycle. Requirements management simplifies and enhances the communication and traceability of requirements (the collection of system capabilities as expressed by the client), collaboration and verification throughout the enterprise.

A poorly expressed requirement can have a domino effect that leads to time-consuming rework, inadequate deliveries and exceeded budgets. A poor requirement can even bring a business out of compliance or even cause injury or death. Requirements management is an activity that can deliver a high, fast return on investment.

Requirements management helps you:

- Ensure that requirements are persistent at all levels of decomposition.
- Assess impact of requirement change to reduce risk.
- Adapt to change throughout all levels of your traceability matrix.
- Incorporate test and QA early in the process.
- Find gaps in traceability.
- Ensure everything is accounted for.

MDSD enables development success

Model-driven systems development (MDSD) is the progressive, iterative refinement of a set of models to drive development of your system. MDSD is straight-forward yet powerful—and it enables extraordinarily complex things to be built from simple pieces. It applies across a wide range of domains and levels of abstraction from very abstract to very concrete, from business modeling to the modeling of embedded software. MDSD is an extension to the Rational Unified Process®.

MDSD is a method for designing large complex systems consisting of workers, hardware and software that consists of a set of transformations that progressively refine our knowledge, requirements, and design. The power of MDSD lies in the power of its abstractions, in its ability to model components of a system.

Why model a system?

We model to manage complexity, to simplify and abstract essential aspects of a system. Modeling enables us to test before we build, to detect errors early and reduce rework, resulting in savings in time and money.

Models help teams communicate more effectively, and can help you integrate and test your products much earlier in the development lifecycle. This reduces costs and increases quality, by enabling you to catch errors earlier in the process when they are much less expensive to fix, and by reducing rework. MDSD uses a set of transformations to iteratively refine your models and your understanding of the system you are building; this helps reduce risk with the increase in knowledge gained through the iterations.

Core issues that MDSD addresses

MDSD addresses a core set of system development problems:

- Overwhelming complexity: MDSD manages complexity by managing levels of abstraction and levels of detail
- Not considering appropriate viewpoints: MDSD provides multiple views to address multiple concerns
- System does not meet functional, performance and other system concerns: MDSD integrates form and function
- Lack of scalability: MDSD consists of isomorphic composite recursive structures and method to address scalability

Benefits of MDSD

MDSD provides many benefits. Some of the more significant benefits include:

- Reduced risk
- Enhanced team communication
- Explicit processes for reasoning about system issues and performing trade studies
- Early detection of errors
- Integration as you go, better architecture
- Traceability

Reduce risk

Many of the activities of MDSD are strictly designed to reduce risk. Models increase understanding, reducing what is unknown, both technically and operationally, so that your technical knowledge increases as you complete iterations. As you produce concrete deliverables, you can better estimate time to completion. Increased levels of specificity reduce the variance in a solution space. By increasing knowledge and reducing variance, MDSD reduces risk.

Enhance team communication

Words can be imprecise. Models can improve communication by making specific a particular aspect of a system. Models can also make system issues *visible* through the use of diagrams—it is often easier to point to a diagram than to describe something in words.

Diagrams can eliminate ambiguity. The very act of modeling or diagramming can force you to be concrete and specific. An MDSD diagram can be printed on a plotter, posted in a central lobby, and become the focal point for discussions about the system across a broad set of stakeholders.

MDSD can also improve communication throughout a development organization. It provides engineers in different disciplines with a unifying language they can use to deal with systems issues. Systems engineers can create models that can be handed to engineers in other disciplines (for example, hardware and software development) to serve as specification for their design. Common use case models can drive system development, testing and documentation.

A common language promotes common understanding. Unified Modeling Language (UML) and Systems Modeling Language (SysML) derive from the same meta-object framework; therefore, products in one or the other are likely to be understandable by multiple disciplines. Cross-organizational discussions are facilitated and use cases, or common system threads, can bring together stakeholders, developers and users around a common vision.

Make explicit the processes for reasoning about system issues

Often, many of our design decisions are implicit—resulting from many years of experience. While this experience can be valuable, if not made explicit it can lead to premature design decisions, or decisions that have not been adequately reasoned through, communicated, tested or verified.

Complexity demands explicit processes. Following a repeatable process improves quality and consistency—and increases our chances for success. In MDSD, process is performing repeatable tasks to produce quality results—in the form of a working system or component of a system.

Detect errors early, when less expensive to correct

A well designed process for developing systems enables early error detection and resolution. The cost of errors rises significantly when discovered later in the system development life cycle or after release.

Trace changes to requirements

Traceability is a common requirement for the systems being built. Often, it is explicitly stated in the contract that traceability matrices shall be provided to demonstrate how the requirements of the system have been implemented and tested. Traceability is also needed to do effective fault or impact analysis—to determine causes for faults and to determine which parts of the system will be affected by a requirements change.

MDSD can help ease the provision and maintenance of traceability information. Three of the core processes of MDSD—operations analysis, logical decomposition and joint realization tables—allow for a great deal of the traceability problem to be automated. SysML provides semantic modeling support for traceability, and the Rational® systems and software delivery platform provides tools and support for traceability.

Components of the IBM MDSD solution

The model-driven systems development (MDSD) solution from IBM includes two industry-leading tools:

- Telelogic® DOORS®
- Telelogic® Rhapsody®

MDSD opens DOORS to a better requirements management process

DOORS is a requirements management and requirements engineering solution that can deliver quality by optimizing requirements communication, collaboration and verification throughout the enterprise. It helps you engineer requirements across disciplines (software, mechanical, electrical) at a system level throughout the product lifecycle. DOORS is a multi-platform system designed to ensure conformance to requirements and compliance to regulations by capturing, linking, analyzing and tracing changes to requirements.

The DOORS component of the IBM MDSD solution enables you to manage millions of requirements and thousands of traceability links, and provides:

- An intuitive document-oriented interface that is easy to adopt
- Comprehensive traceability analysis capabilities that help ensure that no requirement is overlooked
- Change notification that ensures that changes are not missed and that their effects can be thoroughly analyzed
- Integration with popular design, development and test environments that provide complete life cycle traceability

Using DOORS as a component of your MDSD requirements engineering process enables you to:

- Improve collaboration by providing greater visibility of your project goals.
- Respond to changing customer needs effectively and under control.
- Deliver the high-quality systems and software your customers need, on time and within budget.

When it is used with quality management solutions, DOORS can:

- Reduce the risk of overlooked defects, change requests and requirements
- Enable you to test to requirements, rather than just the build, to ensure that you deliver the project according to contract
- Improve process maturity by providing the automated transfer of information—a first step in creating a repeatable process
- Shorten development cycles and accelerates time-to-market by improving project communication, collaboration and information visibility

DOORS provides scalability, traceability, change management and impact analysis, integration with vendor products, and business process optimization to your IBM MDSD solution.

MDSD improves productivity and quality with Rhapsody

Rhapsody is a UML® 2.1- and OMG SysML™-based model-driven development environment for technical, real-time or embedded systems and software engineering. Rhapsody enables the reuse of existing software assets, whether source code or model based. It provides a flexible development environment that enables function-oriented and object-oriented graphical design techniques to co-exist in one environment.

The Rhapsody model-driven development (MDD) environment for real time or embedded systems engineering, software development, and test—based on UML® and SysML—enables embedded systems engineers and software developers to improve productivity, quality and communication by abstracting complex designs graphically, automating the software development process and reducing cost by finding defects through continual testing early in the development lifecycle, when defects are less costly to correct.

Some of the Rhapsody component's key enabling technologies benefit embedded or real-time software developers and systems engineers with:

- A systems and software development environment with complete design portability that supports SysML, UML, DoDAF, MODAF, AUTOSAR and Domain Specific Languages

- Complete application generation for 8-, 16- and 32-bit applications
- Code visualization and reverse engineering
- Integrated requirements modeling, traceability and analysis
- Model-driven testing
- Small- and large-team collaboration

Learn more about DOORS and Rhapsody

To learn more about the DOORS component of the IBM MDSD solution, visit the Telelogic DOORS pages on ibm.com (http://www-01.ibm.com/software/awdtools/doors/productline/). You may also wish to visit the Telelogic DOORS features and benefits site on ibm.com (http://www-01.ibm.com/software/awdtools/doors/features/) or on telelogic.com (http://www.telelogic.com/products/doors/index.cfm). Download the DOORS data sheet from telelogic.com (http://download.telelogic.com/download/article/RAD14037-USEN-00.pdf).

To learn more about the Rhapsody component of the IBM MDSD solution, visit the Telelogic Rhapsody site on ibm.com (http://www-01.ibm.com/software/awdtools/rhapsody/). You may also wish to learn more about Rhapsody's features and benefits on ibm.com (http://www-01.ibm.com/software/awdtools/rhapsody/features/) or on telelogic.com (http://www.telelogic.com/products/rhapsody/index.cfm). Download the Rhapsody data sheet from telelogic.com (http://download.telelogic.com/download/article/RAD14043-USEN-00.pdf).

Why IBM for MDSD?

The MDSD solution from IBM scales successfully from small to large projects. Its treatment of functional requirements, including analysis of the system context, leads to flexible, robust solutions. IBM MDSD enables you to align your products, systems and software development life cycles with business objectives and customer needs to dramatically improve quality and predictability while taming complexity to significantly reduce time to market and overall costs.

PLS03028-USEN-00