Automatic Report Generation in Model-Based Design

Saurabh Mahapatra
MathWorks

ABSTRACT

In recent times, Model-Based Design has been recognized and adopted as a platform for innovation within the automotive industry. The future trends of continued globalization and convergence of the engineering disciplines will pose new challenges for engineers in communicating their design ideas with various stakeholders within their organizations. Automated report generation from software models, rich in information content, can offer a novel solution for addressing these challenges and enhancing process efficiency. We explore the technology in detail with regards to automation, document formats, templates, and interactivity. With examples borrowed from different stages of the development process, we illustrate the types of information that can be captured from a software model as a report for analysis and discussion. We conclude with a set of best practices for reporting that organizations can leverage to strengthen design processes centered on Model-Based Design.

SOFTWARE MODELS AS REPOSITORY OF INFORMATION

For quite some time now, Model-Based Design has been widely accepted in the automotive industry as a development philosophy that has fostered innovation across various organizations while improving their productivity and efficiency. Model-Based Design uses software models to overcome many pitfalls of traditional processes, including communication barriers, manual coding implementations, and building hardware prototypes [1].

Using Model-Based Design as a platform, organizations have accelerated their innovation efforts to meet market and government demands. But to innovate and engineer the complex systems of tomorrow will require a culture of open collaboration. This need can be attributed to various trends that are becoming commonplace – globalization, convergence of multidisciplinary fields, or complex OEM-supplier relationships. One of the key drivers for effective collaboration is sharing information [20]. Sharing information through models is a straightforward solution, although this approach poses some practical hurdles:

LIMITS ON COMMUNICATION – Not all the information contained in the model may be relevant to the issue under discussion. For example, sharing detailed information may not be required for managerial or legal review. One could also encounter a situation in which the OEM may not want to share all the specifics of the model with a supplier.

ECONOMIC VIABILITY – The execution of software models may necessitate the need for many licenses making such sharing economically unfeasible.

WORKING CONDITIONS – It may be impracticable to access software models during calibration or testing on the shop floor.
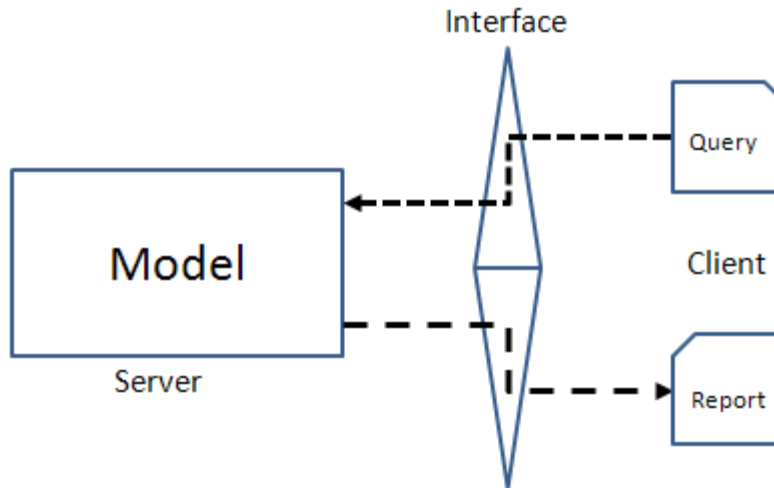
Figure 1: Client-server framework with the software model providing data as the result of a query

Reports provide an easy and efficient means to circumvent these limitations by using views of the model information using electronic document technologies. The information contained in a model takes on many forms, including text-based algorithmic representation, graphical representation of the underlying architecture, parameter descriptors, interface definitions, I/O data, requirements, and associated C/C++ implementation code. This variety leads us to consider a client-server framework [2] in which the model (server) provides reports (service) in response to a query (client), as shown in Figure 1. As we will see in the later sections, such a framework will aid in our understanding of the report generation process.

A CASE FOR AUTOMATIC REPORT GENERATION

The easiest way to query a model for information and capture the results in a report is to do it manually. Therefore, it should come as no surprise that this is indeed the instinctive first line of attack engineers take to tackle this task. In this section, we will describe the limitations of such an approach and make the case for automation.

CHALLENGES OF MANUAL REPORTING – The need for automatic report generation becomes clear to engineers when doing the task manually involves one or more of these following situational aspects.

Scale – The manual creation of reports becomes challenging, especially if the software models have a large number of components. Typically, such an activity would require placing the manual navigation of the model structure and its associated data inside a document. Fixing any errors in the report, especially those discovered late, could potentially raise costs.

Repetition – Even if engineers manage to get past the vagaries of manually documenting large scale models, they cannot escape the needs of a release cycle or running multiple simulations. As engineering is a process of continuous improvement, many of these models will undergo changes necessitating the need for creating the reports again. In simulation-based testing, the engineer may be required to run multiple tests a large number of times and document the results.

Psychological – Engineers have a tendency to shun tasks that are either monotonous or do not utilize their creative talents. With the common knowledge that time is a precious commodity in their day jobs, they are faced with the dilemma of using it for model development, such as adding new features by creating reports manually.

Efficiency and productivity considerations can curb the enthusiasm of various stakeholders to participate in effective information sharing through the use of reports. If the client-server framework, as outlined in

the previous section, was automated by using programmable scripts to query models for information, engineers could then fully invest their creative energies in model development. By doing so, they could avoid the challenges related to scale and repetition while making the best use of their time. The enabling technology for this automation would be an API provided by the software vendor sufficient for gathering complete model information. Note that the concept of automatic report generation has interesting parallels with that of automatic code generation for software models [3].

Query results from software model

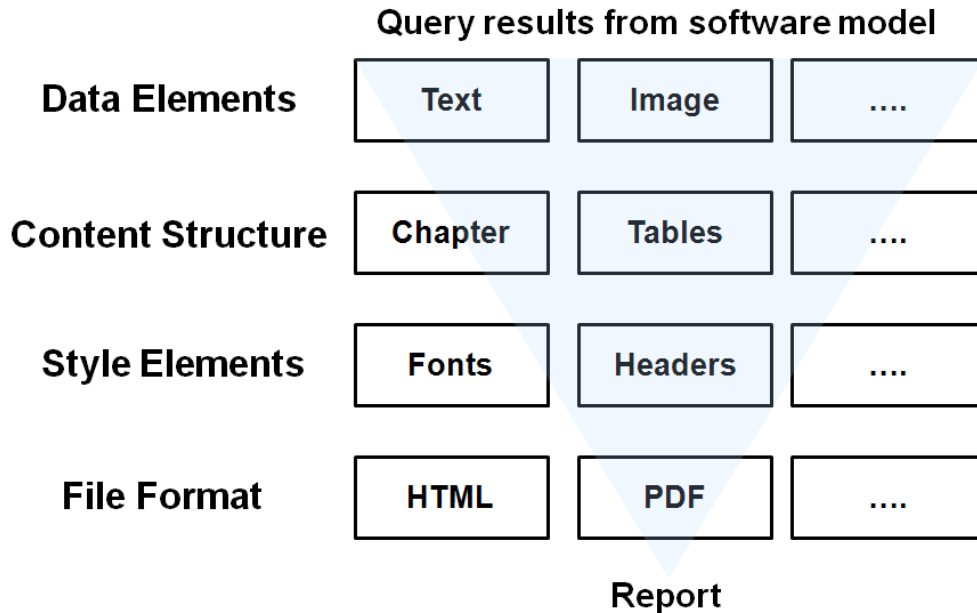|			
Data Elements	Text	Image	....
Content Structure	Chapter	Tables	....
Style Elements	Fonts	Headers	....
File Format	HTML	PDF	....

Report

Figure 2: Fundamental attributes of a report in Model-Based Design. The shaded triangle represents the order in which query results (data) need to be transformed to realize a report.

FUNDAMENTAL ATTRIBUTES OF A REPORT

If we assume that automatic report generation is indeed possible, a question arises: What are the types of reports that can be generated in Model-Based Design? The adoption of Model-Based Design puts no restrictions on the processes that an organization may follow to realize it. Whether the processes involve a waterfall, spiral, or a combination of both [2], artifacts in the form of reports capture the information about the model as it metamorphoses through these stages. For example, from software model to the C code implementation.

Report artifacts share common fundamental attributes that make it possible to classify and understand them [4]. These are data elements, content structure, style elements, and file formats. Let us examine the following attributes shown in Figure 2 in more detail:

DATA ELEMENTS – These are the building blocks or "atoms" of the content contained in a report. These result from queries to the software model. In this paper, we define data elements to be text, images, and interactive elements such as video or hyperlinks. The reporting situation will determine the data elements that will be most predominant within a report. If there is a need for model architectural views only, then the report will be dominated by snapshots of the various components.

CONTENT STRUCTURE ELEMENTS – Content composed of data elements is not useful unless arranged in a meaningful manner. These grouping or structuring elements include, but are not restricted to, chapters, sections, subsections, lists, and tables. Also, the structure of the content inside the report may be dictated by the specific needs of the reporting situation. For example, certification authorities may

require that the reports contain information so that they can ascertain whether the models meet the requirements of their standards.

STYLE ELEMENTS – Presentation of the content plays an important role in how the information is perceived by readers. Style elements such as layout, headers, footers, font sizes, or colors help provide a professional look and feel to these reports.

FILE FORMAT – The final rendering of the report using a document technology also plays a major role in how the information is delivered to readers. Listed below are a couple of well-known formats that could be used to render this information. We discuss specific attributes that are relevant to the main thesis in this paper.

XML – Extensive Markup Language (XML) is a text-based hierarchical format consisting of tags that act as containers for the data elements enabling easy parsing [5]. The need for parsing entails that the hierarchy be defined as a Document Type Definition (DTD). Since this is text-based, the format is popular especially on the web due to its platform independence and smaller file sizes. Despite these advantages, parsing such files can pose readability issues for the reader. Hence, this format is useful as an intermediate format for representing the data information from which it can be translated based on style requirements into HTML, DOC, or PDF formats.

HTML – HyperText Markup Language (HTML) is similar to the XML format with additional tags provided to incorporate style elements that can be rendered by a web browser [6]. The advantages of the format are similar to XML. A limitation is posed by the variability in the rendering behavior of HTML in different web browsers.

DOC/DOCX – This popular format [7] owes its origins to the advent of the WYSIWYG interfaces for authoring documents on the PC platform. The strength of the format lies in the relative ease with which these documents can be created, but they often require platform-dependent renderers for viewing. Also, file sizes are comparatively larger than XML or HTML formats.

PDF – Portable Document Format (PDF) is another popular file format [8] for archiving. Document rendering quality and platform portability are its key strengths. File sizes are comparatively larger than XML or HTML formats. Though free PDF readers are commonplace today, several hurdles still exist for authors to create their documents directly in PDF. Hence, PDF file generation often involves the use of a translator to convert from a file format such as DOCX to PDF.

As shown in Figure 2, these attributes converge to realize a report. The results of a query on the software model provide the data elements that are structured, stylized, and output to a file format for viewing or printing. Despite our understanding of these fundamental attributes, there is one unanswered question: What determines the emphasis and priority on these attributes when generating a report? We answer this question in the next section.

ROLE OF MODEL-BASED DESIGN AND SITUATIONAL DEMANDS IN REPORT GENERATION

Reports are generated in some context within the development process. The context is set by concepts in Model-Based Design and situational demands. Situational demands are concerned with information sharing goals, such as generating reports for certification [9] or managerial decision making. Thus, the utility of a report is determined by how it meets these demands and an understanding of the reporting requirements can help determine the level of effort given to each requirement. For example, if there is need to have a quick review of the model architecture, then the reports generated will lean more toward the generation of high-quality snapshots arranged according to the hierarchy; less emphasis would be given to the style elements. If the report is to be shared through the organization's intranet, then HTML format would suffice.

Model-Based Design Concepts	Situational Demands
Tracing Model/Requirements	Validate requirement early and identify missing requirements or consistency problems.
System Design	Conduct review of system-level design including architecture and parameters with cross-functional teams
Modeling Standards Checking	Identify issues in model impacting simulation and code generation, or maintainability.
Model Functional Testing	Ensure functional requirements are met, preventing rework after implementation.
Model Structural Testing	Ensure completeness of testing, increasing confidence in design.
Model Architecture	Understand system architecture
Tracing Code to Model and Requirements	Ensure consistency among requirements, design, and code
Coding Standards Checking	Ensure coding, especially hand written code, does not introduce defects
Equivalence Testing	Verify design in incremental steps through code generation and running on target

Figure 3: Table showing the example reports based on Model-Based Design with corresponding situational demands.

Regardless of the design process in which software models move from concept to hardware implementation, opportunities exist for reporting that can improve software quality [10]. In Figure 3, we provide examples of these reports, which are based on concepts in Model-Based Design, including requirements linking, design, code implementations, verification, and testing [11,12]. However, their usefulness is determined by the situational demands with stakeholder review.

TECHNOLOGY CONSIDERATIONS FOR IMPLEMENTING AUTOMATED REPORT GENERATION

Implementing automated report generation are impacted by the key ideas that we have discussed so far:

- The client-server framework, in which the model serves as the repository of information and data can be extracted as the result of a query

- Fundamental attributes of reports, including data elements, content structure, style elements, and file formats.

- Concepts for Model-Based Design with situational demands that determine the usefulness of reports.
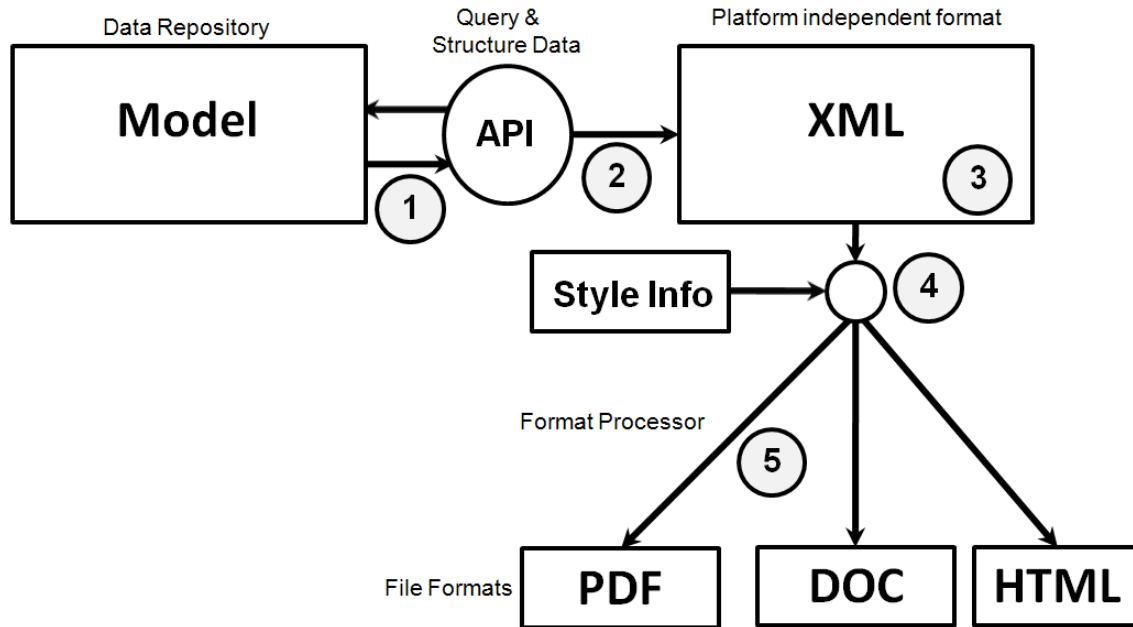
Figure 4: Example implementation of automatic report generation in Simulink Report Generator.

In this section, we discuss one possible implementation of automatic report generation with Simulink Report Generator [13] that incorporates the factors listed above. From a technical standpoint, this example, shown in Figure 4, enables a discussion based on the steps followed to generate reports. In Figure 4, each step appears as a circled number.

The interaction shown in ① is possible through a visual query language for extracting the information from the Simulink model. The language (API) contains elements for both extracting and structuring the data elements, as shown in Figure 5. By nesting these elements in a tree-like fashion, complex content report structures can be developed. The two fundamental attributes of reports – data elements and content structure – can be called as report templates that are executed on the model to extract the information and transform them into XML, as shown in②. The use of templates forms the backbone of the automation because they can be used to generate the reports from models repetitively. Thus, the fixed cost of developing these templates can be recovered by using them across different models and users over time.

The XML generated in ③ conforms to a Document Data Type (DTD) definition given by the DocBook standard [14] for technical publishing. It was originally created by a consortium of software companies as a standard for computer documentation. A major advantage of it is the availability of DocBook tools from many sources, not just a single vendor of a proprietary file format. This intermediate stage is useful because the information extracted from a model is independent of style formation.

If this XML is generated from the model once, post processing onto multiple file formats based on style information results is efficient. Thus, it is advantageous to decouple content creation from the stylizing step.

The style information is programmed as a stylesheet written in Extensible Stylesheet Language (XSL) [15]. The language provides methods for formatting content written in XML. Stylesheets can also be programmed in a similar manner to templates, as shown earlier in Figure 5. Many of these stylesheets are available for free on the Web. Like templates, the fixed cost of developing these stylesheets can be recovered by using them across different models and users over time. The stylesheet and the XML file are inputs at ④ to an XSL processor [16], which is the software that converts the XML into formatted file output at ⑤
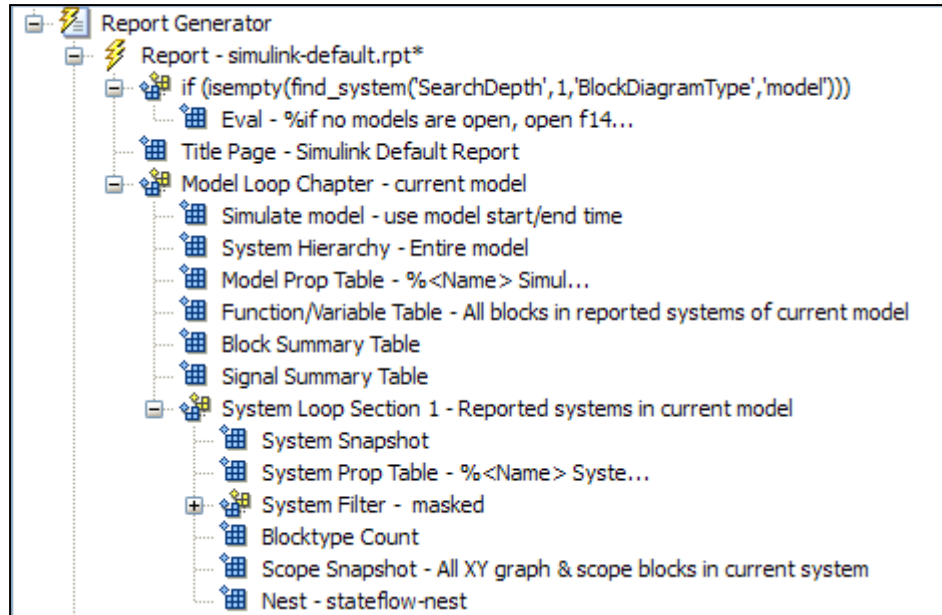
Figure 5: Report content represented as a hierarchy of data and content structure elements. Abstracted reports? such as the Block Summary Table contain both of these elements i.e. block-specific data arranged in a table.

EXAMPLE REPORTS FOR SIMULINK MODELS IN MODEL- BASED DESIGN

In this section, we describe four example reports that embody the ideas we have discussed – fundamental report attributes, role of situational demands, concepts in Model-Based Design, and automation. The purpose of these Simulink examples Simulink is to illustrate the existence of latent opportunities for information sharing in processes centered on Model-Based Design.

SIMULINK WEB VIEW – This HTML report provides a rendition of the Simulink model architecture using Scalable Vector Graphics (SVG) technology [17] for stakeholder review. By clicking through the blocks, users can navigate the architecture of the model in a manner similar to the software environment. Because the images use vector-based representations, information is represented by relationships between elements present in the image. Unlike pixel-based representations, there is no deterioration in image quality with zooming with this format. The function of zooming is an important spatial consideration, especially for understanding large-scale models. Additionally, the SVG standard also contains methods that enable you to interact with the images. As shown in Figure 6, the user can navigate to the *ThresholdCalculation* component by simply clicking on it; component properties are displayed if the mouse pointer hovers over the component.

Because this report is built to meet the situational demand for interactive architecture review through the Web, the style elements are simple. More elaborate stylizing elements can be incorporated using cascading style sheets (CSS).

SYSTEM DESIGN DESCRIPTION REPORT - The System Design Description report [18] describes the system design represented by the model. The content structure of the report is shown in Figure 7 for Simulink models. Note the increased level of detail in the content when compared to a Simulink Web View. This detail reflects the need to support elaborate design reviews and archiving. In Figure 8, we show how style elements such as a header and footer are incorporated on the title page and an internal page that shows a lookup table within the report. Note that the header and footer on the title page differ from those on the internal page. Also, the title page shows a layout with a title, subtitle, author name, and image.
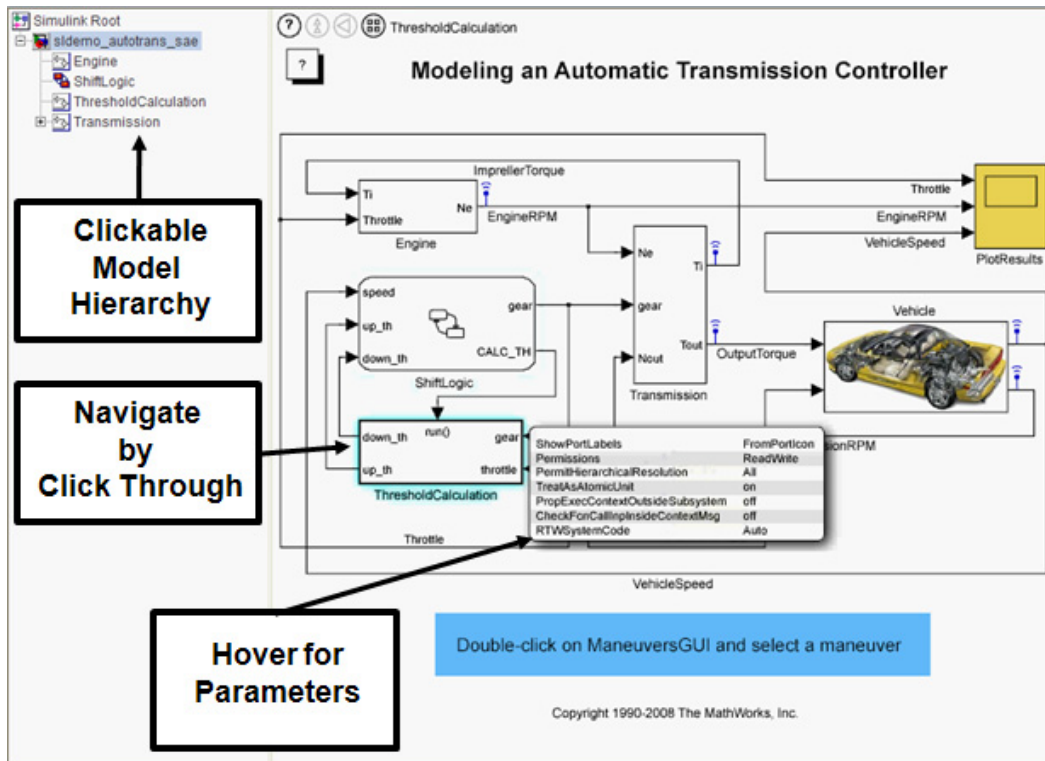
Figure 6: Simulink Web View, in which the model architecture is represented as a hierarchy with the ability to navigate through components and display their properties.

CODE GENERATION REPORT – One of the advantages of Model-Based Design is the use of models to automatically generate C/C++ code and to facilitate traceability between generated code and the source model [19]. The traceability aspect of this report is shown in Figure 9, where the C/C++ code rendered in HTML has hyperlinks that link back to the model and vice-versa.

It is important to note that this report is not passive; it requires the presence of a software model to be useful. The API that we discussed earlier has the functionality that enables this bidirectional interaction. Note that the style elements are minimal like Simulink Web Views.

A similar concept also exists for requirements linking from models where linked requirements appear as hyperlinks in the requirements report. Alternatively, it is possible to create links in the report that trace back into the model [11, 12].

SYSTEM UNDER TEST REPORT – As the name suggests, the content structure of this report is centered around the tests that are being conducted on the model. The input test vectors along with the results of the test are arranged sequentially as shown in Figure 10.

CONCLUSION

In this section, we first summarize the key ideas explored in the paper and then offer a set of best practices that organizations can leverage for successful adoption of automatic report generation as the basis of information sharing among their stakeholders.

Initial discussions in the paper centered on the concept of software models having all the information. Because much of the information may be overwhelming and irrelevant to a step in the development

Section	Information
Report Overview	Model version
Root system	•Block diagram representing the algorithms that compute root system outputs •Description (if available from model) •Interface: name, data type, and other properties of the root system input and output signals •Block parameters •Block execution order for root system and atomic subsystems •Look-up tables •State charts •Requirements •Embedded MATLAB® functions •Model Reference Blocks
Components	Similar information to the information for the root system, as well as: •Path of the subsystem/component in the model •(For atomic subsystems) Checksum that indicates whether the version of an atomic subsystem used to generate the report differs from other versions of the subsystem
Glossary	Software specific definitions of terms

Figure 7: Content structure of the System Design Description Report.

process, reports provide views of useful data that can be reviewed. These discussions led us to a client-server framework in which the model serves as the repository of information and data can be extracted as the result of a query. The benefits of automation were explored in the context of economic considerations such as efficiency and productivity. Next, we discussed the fundamental attributes of reports, including data elements, content structure, style elements, and file formats. The usefulness of a report containing these attributes was determined by two factors: concepts in Model-Based Design concepts and situational demands. Technical considerations along with the above concepts were explored in an example from Simulink Report Generator. Using this example as the basis for reporting on Simulink models, other sample reports were showcased.

The following best practices support the adoption of automatic report generation in Model-Based Design

- *Inculcate a culture of information sharing* – In today's economy–intellectual capital [20], a form of collective brainpower of the organization, is critical for innovation. Managers need to recognize that this capital can be accumulated by encouraging a culture of open collaboration that includes information sharing among all stakeholders.

- *Automate, automate, automate!* – Audit the development process to identify bottlenecks resulting from manual report creation and remove them by using automation. Additionally, free engineer's time by focusing their energies in model development than manual reporting.

- *Encourage engineers to build information-rich models* – Because the model forms the basis of all information, any investment made to improve the quality of the information contained in the model will lead to better information quality in the reports. Engineers should be encouraged to add

useful information to their models as they develop them. Examples of such information include annotations, colors, requirements linking for blocks, and components within the model.

Header Elements

Automatic Transmission System

System Design Description Report

Company Name
Electromechanical Systems

1234 South Hill Rd
Natick, MA 01760

Saurabh Mahapatra

**Title
Page
Layout**

REVISION CODE
86831

DRAWING NO.
9XX-91Y0-E11

REVISION
C

Sheet 1

Automatic Transmission Control System

Table 4.3. Engine Torque

	800.000 0	1200	1.6000e +003	2.0000e +003	2400	2.8000e +003	3.2000e +003	3.6000e +003	4.0000e +003	4400	4800
0	-40	-44	-49	-53	-57	-61	-65	-70	-74	-78	-82
20	215	117	85	66	44	29	10	-2	-13	-22	-32
30	245	208	178	148	122	104	85	66	48	33	18
40	264	260	241	219	193	167	152	133	119	96	85
50	264	279	282	275	260	238	223	208	189	171	152
60	267	290	293	297	290	275	260	256	234	212	193
70	267	297	305	305	305	301	293	282	267	249	226
80	267	301	308	312	319	323	319	316	297	279	253
90	267	301	312	319	327	327	327	327	312	293	267
100	267	301	312	319	327	334	334	334	319	305	275

1.2.1.3. "Integrator" (Integrator)

Table 4.4. "Integrator" Parameters

Parameter	Value
External reset	none
Initial condition source	internal
Initial condition	1000

9X3-9140-9E1 Rev. C
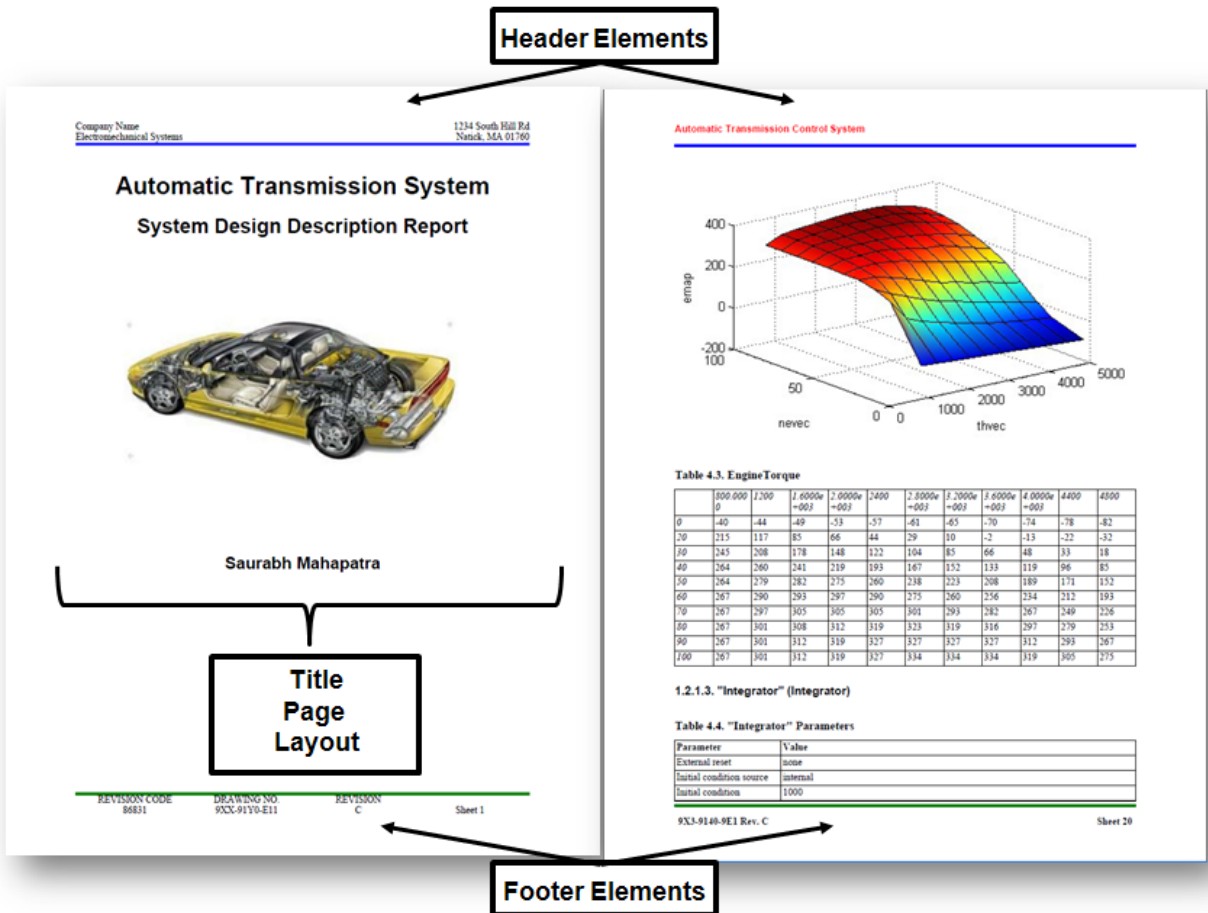
Sheet 20

Footer Elements

Figure 8: System Design Description report with style elements incorporated and rendered in PDF. The title page appears left and an internal page showing a lookup table appears right.

- *Gather requirements for reports* – Identify the types of information that are requested by shareholders. Invest early in gathering the requirements for reporting and decide on the level of investment needed for the fundamental report attributes. Ensure that the cost can be recovered by building templates and stylesheets that can be run either on large models or multiple times to generate reports.

- *Integrate reporting as part of the development process* – Models will undergo numerous revisions by several engineers during their lifecycle. By generating reports at checkpoints in the development process, a snapshot of these changes happening to the model can be used later to diagnose issues or enhance features.

- *Implement a content management system* – As the number of reports increase in the system [21], it becomes essential to build and manage a repository of reports. Using a content management system with features that range from tagging reports to revision control..
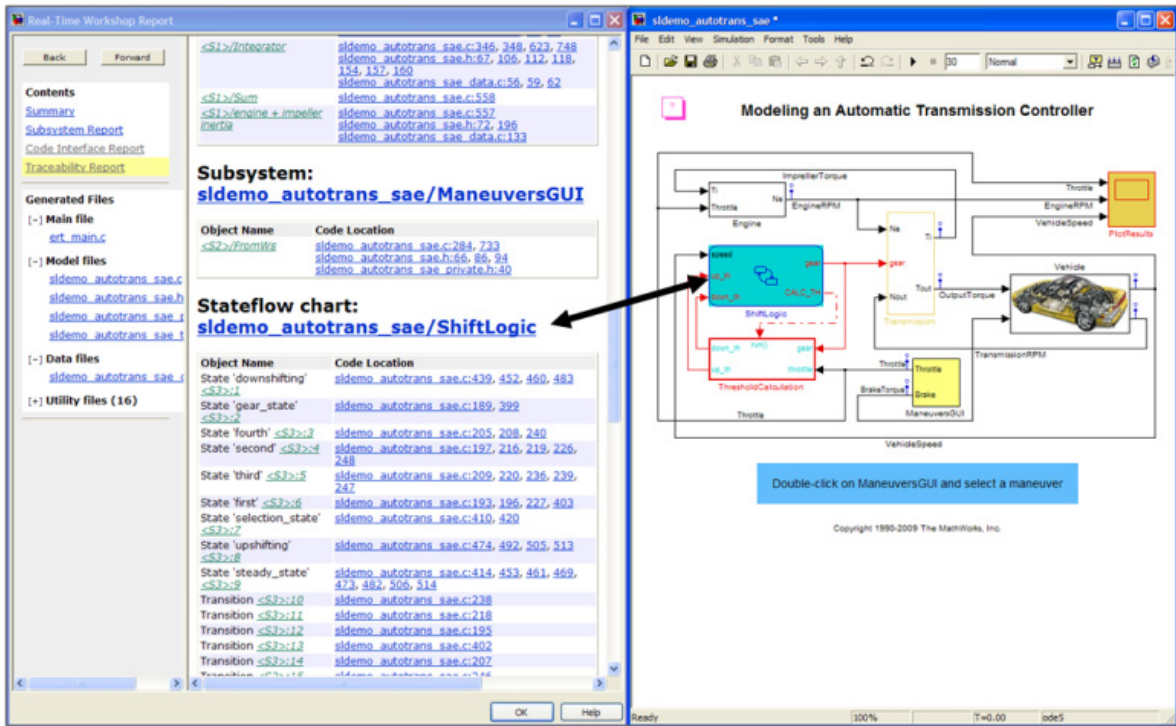
Figure 9: Code Generation Report featuring traceability that links the code to the source model blocks and components.
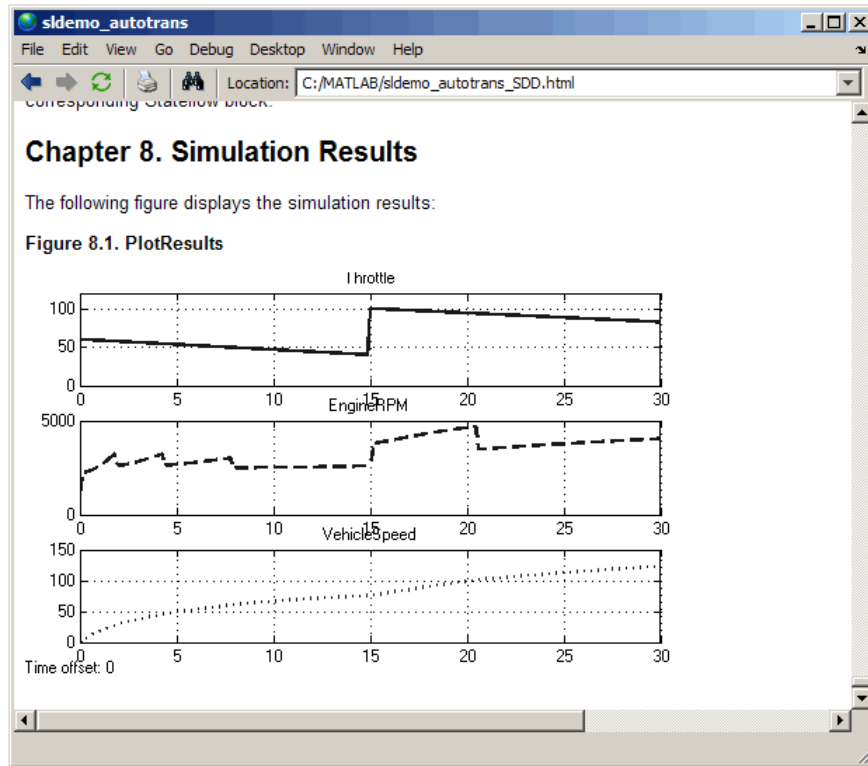

Figure 10: System under test report displaying the simulation results- vehicle speed and engine RPM for a throttle input to an automatic transmission controller.

ACKNOWLEDGMENTS

REFERENCES

1. S. Mahapatra, T. Egel, R. Hassan, R. Shenoy, M. Carone, "Model-Based Design for Hybrid Electric Vehicle Systems," SAE Paper 2008-01-0085.
2. C. Ghezzi, M. Jazayeri, D. Mandrioli, "Fundamentals of Software Engineering," 1st Edition, Prentice Hall, 1991.
3. T. Erkkinen, S. Breiner, "Automatic Code Generation—Technology Adoption Lessons Learned from Commercial Vehicle Case Studies," SAE Paper 2007-01-4249.
   K. Shriver, "Dynamics in Document Design: Creating Text for Readers" — 1st Edition, 1996 Wiley.
4. World Wide Web Consortium (W3C) – Extensible Markup Language (XML), <http://www.w3.org/XML/>.
5. World Wide Web Consortium (W3C) – HTML Specifications, <http://www.w3.org/MarkUp/>.
6. International Standards Organization (ISO) – Open Office Open XML specification, ISO/IEC 29500: <http://www.iso.org/iso/pressrelease.htm?refid=Ref1181>.
7. International Standards Organization (ISO) – PDF format becomes ISO standard : <http://www.iso.org/iso/pressrelease?refid=Ref1141>.
8. M. Conrad, G. Sandmann, J. Friedman, "Verification and Validation According to IEC 61508: A Workflow to Facilitate the Development of High Integrity Applications," SAE Paper 2009-01-2929.
9. P. F. Smith, S. Prabhu, and J. Friedman, "Best Practices for Establishing a Model-Based Design Culture," SAE Paper 2007-01-0777.
10. T. Erkkinen, M. Conrad, "Verification, Validation, and Test with Model-Based Design, SAE 2008-01-2709.
11. M. Conrad, I. Fey, H. Dorr, I. Sturmer, "Using Model- and Code-Reviews in Model-Based Development of ECU Software," SAE 2006-01-1240
12. MathWorks Inc., Simulink Report Generator – Documentation: <http://www.mathworks.com/access/helpdesk/help/toolbox/rptgenext/>.
13. N. Walsh, R. Hamilton, "DocBook 5: The Definitive Guide," 1st Edition, 2010 O'Reilly Media.
14. B. Stayton, "DocBook XSL: The Complete Guide" — 2nd Edition, 2003 Sagehill Enterprises.
15. World Wide Web Consortium (W3C) – XSL Transformations, <http://www.w3.org/TR/xslt>.
16. World Wide Web Consortium (W3C) – Scalable Vector Graphics (SVG) 1.1 Specification, <http://www.w3.org/TR/SVG/>.
17. Institute of Electrical and Electronic Engineers (IEEE) – IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions, <http://standards.ieee.org/reading/ieee/std_public/new_desc/se/1016-1998.html>.
18. B. Chou, S. Mahapatra, "Techniques for Generating and Measuring Production Code Constructs from Controller Models," *SAE* International Journal of Passenger Cars - Electronic and Electrical Systems, October 2009.
19. T. Stewart, "Intellectual Capital: The New Wealth of Organizations," 1st Edition, Broadway Business, 1998.
20. A. Rockley, "Managing Enterprise Content: A Unified Content Strategy," 1st Edition, New Riders Press, 2002.

CONTACT

Saurabh Mahapatra, Product Marketing Manager, Simulink Platform Marketing, MathWorks.
Saurabh.Mahapatra@mathworks.com