Abridged Edition:

A Case for Model-Based Architecting in NASA

Bob Rasmussen

Brian Muirhead

August 2012

Preface

Model-based architecting is a structured approach to system architecting, intended to address many of the difficulties projects have experienced in managing complex developments. The intent of this summary-level white paper is to provide basic background and information on the role of architecting on NASA flight projects, the deficits in its present practice, and the motivations and value for a *model-based* approach to architecting.

This is not a how-to manual, but rather a holistic treatment of what architecting should be and why it is important. Included are unfolding ideas on how to advance the discipline, and examples of key concepts. Since the concepts described here may be unfamiliar, and putting them into practice is still in its formative stages, readers are encouraged to seek out and collaborate with other practitioners of these ideas to understand and advance this important element of systems engineering.

The emphasis here is strongly on flight projects, and largely on the systems engineering and management aspects of these projects. This treatment of model-base architecting is intended primarily for systems engineering and management leadership (both programmatic and technical) at all levels of project development, who are interested in this approach and the value it might have in improving the way we conduct NASA business.

The Issue

A recurring theme in any assessment of NASA projects is the problem of poorly conceived or executed system formulation, and the consequent harm on cost, schedule, reliability, and performance during implementation and operations. A key component of this issue is uncontrolled complexity in what we build and how we do business. Complexity is not uniquely a NASA issue, but as a highly public enterprise with exceptional demands, a unique look at the complexity issue from a NASA point of view is very much in order.

Complexity is also not a new problem for NASA. It is intrinsic to our mission, and we have diligently worked through such issues before, learning from each experience. Nonetheless, the general unease today is that we may have arrived at a significant point of departure, where the weight of current practice may itself be a hindrance to progress in the face of increasingly difficult missions. The problem now seems to be less about overcoming technical challenges than about defending our ability to affordably, reliably, and consistently manage all of the necessary ingredients for success. That is, these problems are both technical and programmatic.

Technical and Programmatic Concerns — The essential assertion behind the recommendations here is that the essence of today's complexity issue lies in a disconnection *between* technical and programmatic concerns. Development and operations problems across the agency highlight the interrelationship of technical and programmatic factors, and suggest an inability to fully appreciate or reconcile all of the competing demands on a project.

There has been a tendency to levy this criticism mainly at the *technical* aspects of a job, with programmatic repercussions viewed merely as a symptom of technical challenges from an overly complex system. In reality though, only the interplay *between* technical and

programmatic factors can tell the whole story. Besides, technical responses to increased demands are inescapably additive anyway (harder objectives demanding more complex responses), so increasing technical difficulty is consequently a given, not an excuse. This is central to the nature of NASA's task; a conundrum NASA is expected to solve.

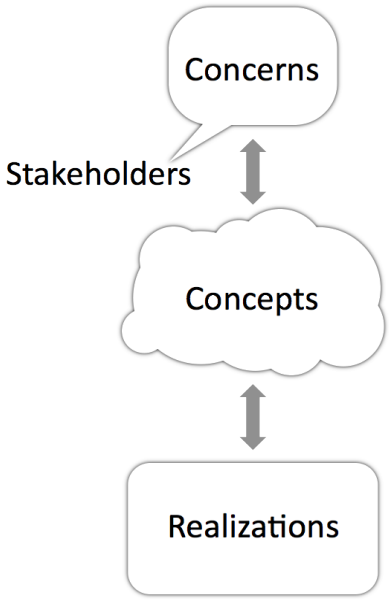
The necessary insight then, to ensure further progress, is acknowledgement that complexity is not a system property existing in dispassionate isolation. Rather, since what is well understood does not generally seem overly complex, we must see complexity as fundamentally the comparative measure of a system (the technical part) against the grasp of those who must understand and control it (the programmatic part).

A Question of Understanding — From this perspective, it becomes clear that affairs dominant in programmatic management are all part of the complexity question: stakeholder objectives and constraints, feasible mission concepts, communication and review, work breakdown and costing, principles and policies, validation efforts, and so on all directed at ensuring a comprehensive understanding of the whole enterprise and the means to manage it. Thus, the need to ensure sound understanding makes complexity (the measure of our understanding) necessarily a property with inseparably coupled technical and programmatic dimensions.

Problems of understanding have many roots — mostly sins of omission by bright, well-intended people. Thus, what one should imagine about lapses in understanding is not a perplexed team, scrambling in disarray, but rather a confident team, acting blithely upon incomplete understanding, and unaware that something is missing. By the time problems are discovered, a project can be well into development, or even beyond, making resolution a very expensive proposition. Our concern here then is how to better ensure that a comprehensive, well-integrated, consistent understanding of a system will be *assembled early* and then *maintained intact* throughout development. The word generally applied to this process, which we are trying to reinforce across NASA, is "***architecting***".

The Meaning of Architecture — Architecting is a broad term for the approach we take to get from system objectives and constraints to design. It must be recognized as an exploration, both creative and systematic, of essential concepts and organizing principles, addressing a typically intricate, often evolving variety of concerns, and leading to convergence upon a mediated, realization that is harmonious (or at least acceptable) from all points of view. Just as importantly, this "architecture" provides unwavering assertion of these carefully crafted ideas as durable, principled guidance for the remaining development.

The cardinal rule of architecting, therefore (as advocated here), is to keep in mind that the product of architecting is not a system! Rather, it is an *understanding* of the sort of system that needs to be built



Concepts are essential abstract ideas, formulated to address **Concerns**, but taking implementable form in **Realizations**.

and our expectations for it, with all their compromise — as much a matter of *why* and *how* as of *what*. Understanding exists *apart* from the system it addresses.

Viewing the architecture as separate from the system is fundamental and provides illuminating insights. Poor architecture helps explain the complexities of delivering systems, not because the architected system itself is necessarily complex, but rather because the architecture fails to provide a good understanding of what the system should be or do. Good architecture, on the other hand, carefully establishes the context of a system through its purpose, efficiently applying principled concepts and patterns that support effective design and analysis, and openly dealing with all challenges to robustness.

This is the essence of architectural *elegance*, all with the aim of fostering understanding in service of a successful design outcome. As a separate entity, elegant architecture becomes the persistent, stable conceptual rails over which a system travels on its way to realization. This notion applies from the broadest enterprise level (like all of human spaceflight), to single missions, engineering subsystems, and beyond.

Commitment to Elegance — Appreciation of the steps to elegant architecture requires a closer look at current architecting practice, to explore vulnerability to technical and programmatic disconnects, to consider principles of architecting that must be elevated in order to confront these issues, and to build from this basis a strategy for architecting, better suited to managing future missions, especially those of high complexity.

The recommendations suggested by this assessment involve the methodical, model-based structuring of a unified *architecting product*, distinct from design and deserving the same stature in management attention as any other major end product. The resulting architecture is shaped to facilitate timely, balanced attention to concerns, relationships, and the organizing principles that promote understanding. Primary components of this approach are 1) the adoption of a principled *architecture framework*, and 2) the formal integration and application of this framework in a *model-based systems engineering* methodology. The result is *model-based architecting*.[1]

A Look at Present Practice

Model-based architecting should not be viewed as yet another addition to present practice; nor is it intended to displace it entirely either. The changes envisioned can be accomplished in a fairly straightforward manner. With this in mind, a few observations regarding present practice are in order to indicate areas for enhancement, and to illuminate some of the principles that should inform this recommendation.

Architecture in NASA SE Processes — As defined here, architecting is a key element of systems engineering.[2] For all its importance though, formal guidance for architecting in

[1] For an excellent introduction to formal frameworks and their model-based representation, refer to ISO/IEC/IEEE 42010:2011, *"Systems and software engineering — Architecture description"*, an inclusive international standard for architecture frameworks. Access links may be found on the ISO/ IEC/IEEE 42010 website.

[2] While this could as easily have been described as a project management responsibility, supported by systems engineering, addressing this collaboration from the systems engineering viewpoint is more straightforward.

NASA policy and procedure is nonetheless subsumed within a larger systems engineering model, within which architecting is not given a clearly delineated role, and where the products of architecting are somewhat fractured or indistinct. Architecting is not even considered in NPR 7123.1A, *NASA Systems Engineering Processes and Requirements*, and most other NASA procedures.

More promisingly, the *NASA Systems Engineering Handbook* (NASA/SP-2007-6105 Rev1) describes more fully an approach that includes many aspects of system architecting. However, the words "architect", "architecture", or "architecting" casually appear many times before "architecture" is defined on page 50. The ideas of structure and relationships are introduced, and subsequent discussion mentions the importance of "principles and guidelines" that govern the design. Still, little overt attention is given thereafter to what "principles and guidelines" might mean in practice, and even the organizing structure of an architecture is generally treated in summary terms.

Generally, one gets the impression that most SE Handbook invocations of the word "architecture" intend only narrowly to define one product breakdown hierarchy of interconnected components and overlaid functions — with little distinction between "design" and "architecture". The broader aims of architecting, as outlined earlier, are addressed (for the most part). However, this is almost always woven indistinctly into the broader banner of "systems engineering", not "architecting" *per se*.

Architecture appears to be treated more as an occasional descriptive device than as an essential product on its own. The system and its architecture are not distinguished. Systems engineering and architecting are not distinguished. Therefore, because *The Architecture* is not a distinct product, easy for everyone to grasp completely, its integrity becomes suspect and its influence can be weak or inconsistent — not a sure-fire formula for achieving elegance. This is the core issue we intend to address.

Seeing Beyond Requirements — The subsumption of architecture into requirements generation is a prominent example of this lack of appreciation for the separate nature and value of architecting. The NASA SE Handbook does a fine job of laying out the ingredients of good architecting. However, in essentially every case, the result is expressed as either requirements or design.

The familiar "flow down" of requirements in unfolding levels of design detail bottoms out when system decomposition is complete, each requirement supposedly accompanied by its own isolated statement of rationale. But no concatenation of requirements and rationales ever sums to a full, coherent story of architecture. Further, the practice of tracing requirements directly to other requirements can miss other important considerations that aren't evident in requirements-oriented thinking. Ultimately, the resulting requirements — not architecture — dominate subsequent discussion, as attention shifts to putting the system back together again. Design, implementation, integration, verification, and so on all revolve around requirements.

Assorted supplemental information (presentations, emails, memos, etc.) tends to present only a fragmented, cursory, and not altogether dependable, accessible, consistent, or up-to-date rendering of the architecting effort. Thus, the collection of requirements is one of the few sources of architectural insight for which one official copy is maintained, fragmented as

it might be. Requirements alone, however, cannot be relied upon to address the full range of obligations and constraints under which a project or system works, or the concepts responding to them. Further, efforts to create a comprehensive picture through requirements would result in an overly complex set of requirements, an inappropriate degree of over-specification (thereby inhibiting development creativity and flexibility), misdirection of attention from the big picture, and increased difficulty in the verification process.

In such a diverse, disjoint, diluted collection of architectural artifacts lies vulnerability that the form ultimately taken by a design falls substantially short in the eyes of some (perhaps many) stakeholders. This is a recurring theme in criticisms of systems engineering, where omissions, misinterpretations, inconsistencies, and poor foresight regularly lead to problems later in development or during operations.

Transcending Design — The absence of an overt appreciation for architecture as a distinct product also becomes evident wherever we see "high-level design" masquerading as "architecture". Architecting necessarily confronts disorder in the early stages of a project, when many issues remain open. However, the unease associated with exploration of competing demands in a broad trade space is often dealt with by pushing rapidly for a point design that *seems* to work, well before a thorough understanding of alternatives and threats. Having rationalized this approach in the name of "decisiveness", the process thereafter becomes a progression of today's design (occasionally reviewed), not architecting. One is left with significant uncertainty and risk that the system can and will ultimately be coerced into convergence while meeting its objectives within constraints.

The extent to which this fate is avoided often depends on all-seeing veterans with the architectural sensibilities to instill by insight what was absent in foresight. However, most systems have simply gotten too complex to rely upon this as a strategy for success, and the cost of late correction can be enormous. It is far better to take the time for architecting and the care to do it well, by encouraging everyone to look at systems holistically.

Rediscovering the System (aka V&V) — When the big picture gets lost during development, this does not mean that the system disappears. Rather, what goes missing is an *understanding* of what this system will be once it is integrated (un-disintegrated?). This rediscovery of the system (though usually explained otherwise) is called "validation".

Validation is a rather incongruous idea, given the heavy emphasis on requirements in systems engineering processes. The flow down of requirements and iterative climb back up again through requirement verification essentially defines the 'V' model life cycle. Nonetheless, the NASA SE Handbook asserts that additional "expectations (e.g., needs, wants, desires, capabilities, constraints, and external interfaces)"[3] *beyond* the requirements are essential. Validation of end products, either concurrent with or *after* verification, addresses these expectations. This notion is carried into most institutional practice across NASA.

Underlying such expectations for validation is unmistakably an acknowledged likelihood that the system emerging after verification may not be the one intended at the outset, despite satisfying all the requirements. Thus, the purpose of validation, at least in part, is to discover what system actually got built! As defined, it is subjective and open-ended, and

[3] See SE Handbook section 4.1.1.3 "Outputs" (of the Stakeholder Expectations Definition Process).

can call a design into question after it's been built. Such devolution of understanding is not uncommon, but its implications can be dramatic, dangerous to mission success, and expensive.

Verification may have deficits as well, such as when testing becomes a means of design iteration, or when test success is equated with system correctness. Careful verification methods derive their merit, not from test results alone, but also from the association of test results with architectural assertions about why these results were expected and how these results can be extrapolated to all reasonable situations that could *not* be tested.

Architectural assertions then are effectively the theory of a design. These can be as common as good margins or sparing plans, or as particular as a graceful technology exit strategy or an error-tolerant interaction protocol. They can be anything aimed at simpler modeling and analysis, such as layered designs or symmetric configurations or uniformly applied standards. They can be the assertion of models themselves, as the spanning definition of system behavior. Thus, they go beyond assumptions and point design, to overtly describe the features and constraints imposed upon a system for efficient flexibility or robustness; and on this basis, they explain one's *understanding* of why the system should be or do the right thing in all relevant cases. They justify inherently finite test programs!

It follows that verification tests should be more like experiments to support or invalidate the theory, to explore its generality, to find it limits. Minus an invocation of this reasoning, verification can devolve to a perfunctory, ineffective checklist exercise.

Reasserting Architecture — To accomplish such things convincingly, the underlying architecture must follow straightforward patterns of structure and behavior that have been carefully described and communicated, that are evident in the design, and that have been implemented without exception or compromise! Absent such patterns for each area of concern, understanding suffers. This is Occam's razor for Architecture. It is the existence of such architectural assertions that offers some assurance of sustained system understanding during development, and eventually a confident extrapolation from test results to flight behavior. These assertions *are* the big picture: the fundamental ingredients of architectural elegance.

By aiming V&V at our *understanding*, it is clear that these ideas apply to the confluence of technical and programmatic interests. Architectural assertions, after all, necessarily cover a broad range of issues beyond the attributes of delivered end products, and should be subject to V&V as any other claim about the system would be. This is the power of unifying efforts around a few fundamental architectural notions. The sensibilities gained in one area become extendable to other aspects of the endeavor; easier methods of relating competing concerns arise when structure is shared; and diffuse terms (like validation) can begin to gain some solidity. All of this improves and preserves understanding.

Architecting Principles

To improve the architecting process by structuring its practice, architecting must be distilled to its essence — complex explanations for complex processes being of little value. This idea is the genesis of so-called *architecture frameworks*, mentioned earlier. The structure selected should be in accordance with basic principles. The following principles,

discussed above, apply to the general practice of architecting and to architectures as a whole:

Architecting is a direct response to complexity: the measure of how well a system is understood by those concerned with its development and use.

Architecture is developed to promote an early and sustained consensus of system understanding from many technical and programmatic points of view.

Architecture is not merely high-level design, but rather provides the coherent rationale from which requirements (and other directives on the development effort) are drawn.

Elegant architecture demands steadfast attention to purpose, understandability, and robustness of the system *as a whole*, even as the details develop.

Architecting works best as an overtly distinct and sustained effort.

Good architecture provides stable guidance to development by getting the fundamentals right and asserting them steadily and consistently.

The integrity of a system lies, not merely in passing tests, but in demonstrated adherence to solid architectural assertions.

Additional principles addressing architectural content may be added, as follows. These are not intended to be a complete set of principles, but they are important on their own, and are offered to give insights into the nature of other good principles that might be added:

Usefulness and acceptability lie in the association of systems with stakeholder concerns, so architecture must cover both, and do so thoroughly and uniformly.

> Concerns, in this respect, take many forms, depending on the stakeholder (e.g. science objectives from a Principal Investigator, budget constraints from Program Management, and so on). No meaningful description of a system is possible without this context, so architecture necessarily deals with how the system relates to the stakeholders who drive it. Dissociated requirements aren't enough.

Architecting defines a space of designs by allowing for variation (margins, contingencies, etc.) and defining what variations are allowed (rules, patterns, etc.) across the full spectrum of concerns.

> Point designs fail to illuminate the sources or implications of inevitable change, whether during development or afterward, such that the tolerance of an architecture to this variation is understood. Margins are an example of addressing variation, but by no means cover all variations of concern. In other cases, variation must be explicitly constrained (as in configuration symmetry or uniform interface standards) in order to impose understandable order. Thus, architecting the space of designs is effectively the art of choosing, out of all the designs that are possible, not *the* design, but rather a subset of designs that is small enough to establish order, but large enough to hold a set of options that makes response to problems possible.

Good separation of concerns requires overt attention to concepts, as a way to ensure that all issues get proper attention, and to map concerns to realization in an understandable and modelable way.

Modeling generally involves abstraction into conceptual forms suitable for analysis (dynamic, thermal, operational, etc.), where details incidental to the issue at hand are disregarded. Each issue can then be considered separately with confidence, as long as the underlying realized system maps cleanly to these abstractions. Architecture can promote such understanding by recognizing these conceptual views as *driving* the realized system, rather than being derived from it, and as the *direct* response to stakeholder concerns, rather than as merely a post facto affirmation of them. For example, the conceptual approach to redundancy and fault containment should be in harmony with concerns over safety and reliability *before* concrete design choices are made, and then preserved explicitly throughout development.

True logical decomposition avoids a dominant hierarchy in favor of strong, distinct structure for each concept, within which functions and other conceptual features are defined.

The compositional elements of an electrical grounding and isolation concept are clearly distinct from those of thermal management, data flow, or other concepts. Yet requirements hierarchies tend to be defined mainly by work breakdown and modular integration concepts. Such conceptual misalignments tend to clutter requirements and obfuscate their rationale.

Conceptual and realizational efforts should work in partnership to achieve conceptual integrity and support realization requirements with robust rationales.

Individual concepts alone may be attractive, but impractical or incompatible with other concepts. As a result, engineers are drawn pragmatically to consider hard realizations. However, realizations alone convolve all issues at once, not always supporting good separation of concerns. This makes good understanding difficult, if not impossible. A balanced interplay and co-convergence between these two realms of architecture permits the principled evolution of a compromise that can stand the test of time. This occurs, for instance, when the realizational specifics of data interfaces are guided by system-wide concepts for data flow, fault containment, electrical isolation, and so on, which in turn respond to the practical matter of implementation through appropriate adjustments.

Understanding the intersection of concepts in realizations is a powerful way to trace and understand architectural dependencies.

Great pains are taken to specify and characterize interfaces between realized components (as captured, for instance, in interface control documents), but dependencies among concepts, which are the consequence of their mutual realization in one system, seldom get the same formal treatment. One could ask, for example, how a concept for electrical power influences a concept for precision pointing through the shared realization in solar panels of both power generation and pointing dynamics. The realized solar panel is effectively an interface *between* power and pointing concepts rather than merely the target of requirements *from* each concept.

When all concerns are addressed in collaboration, and mapped through well-separated concepts to realization, architectural layering is much cleaner and easier to understand.

Requirements are frequently "decomposed" along presumptive subsystem lines, so architectural layers may not be well aligned with all the concepts that inform a particular architecture. The result can be over-specification and awkward, strained tracing of requirements. Architectures are most elegant that achieve their aims in the simplest, most efficient terms. The realizations resulting from reconciliation and integration of minimal concepts will themselves be minimal, while nonetheless going to the depth and granularity appropriate to each concern; so, requirements derived from this convergence define a natural layer in the architecture from which the next round of elaboration can proceed. Requirements trace not directly, layer to layer, but rather through a reasoned mediating architecture between them.

Adding Structure

If we are to shift current practice toward a more structured approach that helps us manage complexity by keeping important issues clearly in view, it will be necessary to devise a structure that inherently reflects these principles.

Formal architectural structure alone is not enough to define good architecting practice. Rather, the motive behind formal architecting structure is that architecting practice benefits from structure's contributions to clarity and coherence. The regular structure we choose must be straightforward and concise, while nonetheless aptly covering the many dimensions of architecting that we care about. Besides being simple, the structure adopted must also be clearly articulated and rigorously applied, so it can guide architecting with some measure of assurance. The idea of an architecting structure, after all, is to build a network of related information, where items are clearly delineated, found in logical places, and connected in helpful ways, all aligning well with the principles we understand to be important.

Words like stakeholder, concern, concept, realization, function, property, and so on are good candidates for the atoms of this structure — a lexicon drawn from familiar systems engineering terms, but not yet molded by particular rules for how they are to be applied. What we intend, in order to formally structure the architecting process, is to complete this list and more carefully articulate what it means to describe such things and relate them to one another.

Objectives and Organization — To establish effective objectives regarding the capabilities and scope required of a model-based architecting framework, it is necessary to acknowledge and embrace the fundamentally different natures of information-as-communication versus information-as-knowledge.

As knowledge, one can think of architecting structure as the beginning step in constructing an "encyclopedia" of architectural information, providing the form within which a curated description of architecture can be assembled, organized, and maintained. The result of populating this structure is a compilation of essential information, categorized and cross-referenced in an orderly fashion for quick access with minimal duplication — a **Single Source of "Truth"**[4] from which all architecting information flows.

[4] This is common IT parlance, referring not to some privileged origin of Truth, but rather to reliance on one authoritative source for every item of information.

As communication, the structure must also be amenable to direct extraction of content suitable for narrative description. Thus, it must support intersecting hierarchies and easy cross-referencing, permit partitioning that avoids duplication, make room for both narrow specialized content and general summary content, be formally specified (hence, amenable to automation, distributed collaboration, and controlled access), have a granularity appropriate for archiving changes, and include a way to distinguish what is particular about an architecture from what are generic, reusable organizing patterns. Features like these, beyond simply aligning with good architecting principles, are necessary in order to build a usable architecting structure.

Transitioning to Model-Based Methods — In NASA's document-centric world, the dominant state of current systems engineering practice is the collection of information in stacks of viewgraphs accumulated (with supporting spreadsheets, graphics, and a few other things) in weakly organized file systems (now network-accessible so all can contribute to its disarray), metadata frequently consisting of little more than cryptic file and folder names (defining data by *where* they are, not *what* they are). The full reports that *are* written further the replication of information in a variety of forms; and the few tools that support collaboration fail to address information integrity due to their individual limitations in telling the whole story and their lack of functional and transparent connections among one another. As a consequence, systems engineers often devote a great deal of their time merely serving as the human ties among all of this data, too often at the expense of timely, creative, thoughtful, analytical efforts. Given this bureaucratic state of affairs, there is general consensus that more integrated collaborative capabilities are needed. However, all-purpose information repositories generally lack an a priori structure suitable for systems engineering, and superimposed ad hoc structures are hard to establish and maintain (nor can they, by their very nature, provide a versatile, long term solution).

Tools created explicitly with systems engineering in mind have made significant headway, many employing the Systems Modeling Language (SysML), an open standard defined originally in 2003 by the International Council on Systems Engineering (INCOSE). Still, all but the most elementary structure is normally left up to users, who are expected to bring methodology and commensurate additional structure to the tools. Importantly, only with such additions does model-based systems engineering start to acquire real effectiveness. Beyond this level are tools that reflect a vendor's particular methodology. While not necessarily supporting the principles of architecting advocated here, they do reflect methodologies in wide use, and can help put engineers further down the path toward true model-based systems engineering. Nonetheless, it is also clear, *especially* in regard to architecting, as outlined here, that no firmly established pattern yet exists — let alone one fully suitable to NASA. Indeed, it isn't apparent yet whether it is appropriate to even settle on a single pattern that would cover our wide range of project types.

The goal here is not to weigh all of these alternatives and try to answer this question now, but rather to describe the sorts of basic structure one might consider in experiments to see what works best. With support from NASA engineering leadership, this experience can grow into an organic yet effective process, leading eventually to agency-wide capability.

The following is from one such trial, presently under way on Europa mission studies. This structure is not intended to be definitive; only the simplest and most obvious structural ideas are covered. But this has turned out to be enough so far.

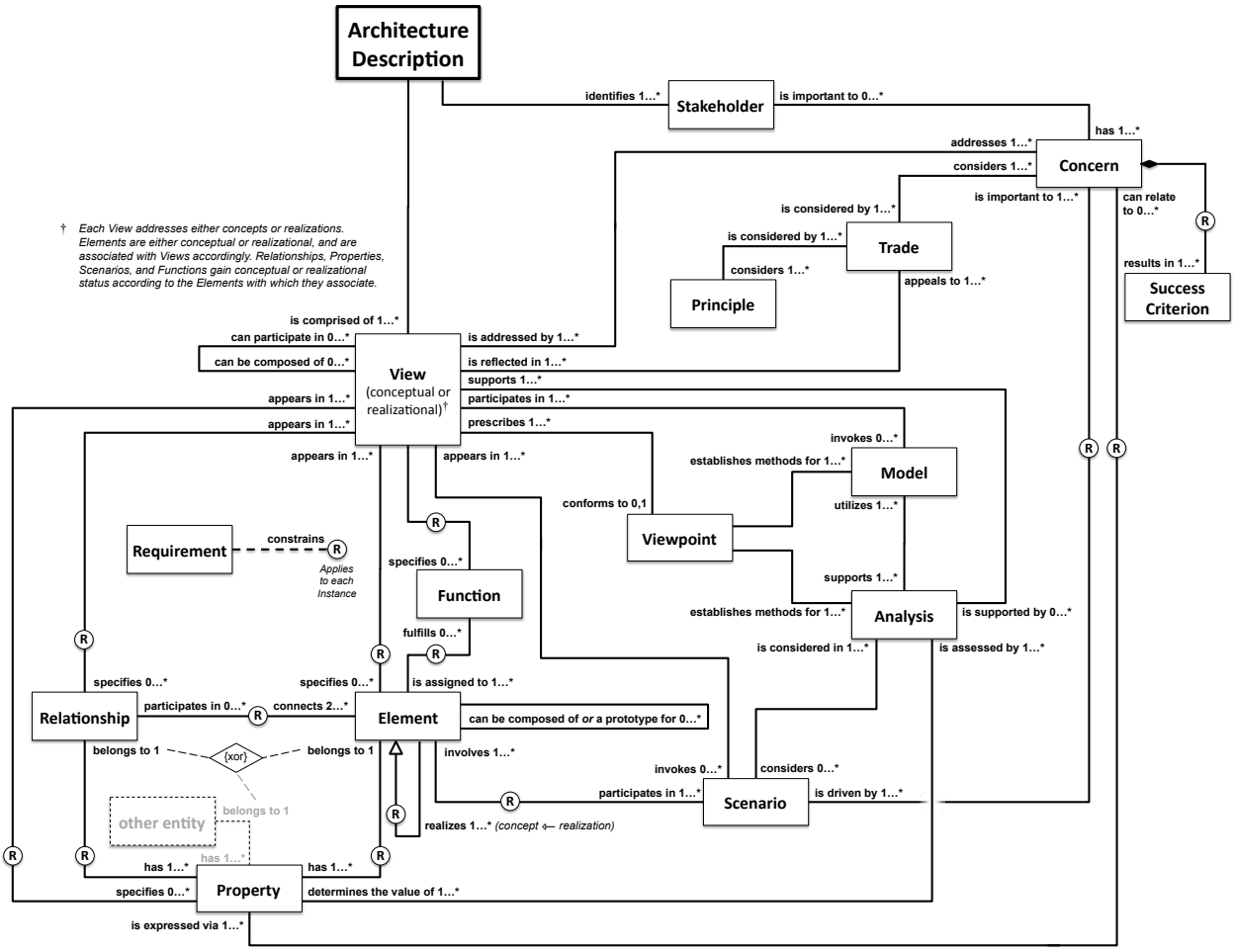
Basic Architecting Terminology — Some of the more common terms from architecting literature are as follows:

> **Stakeholders** determine whether a system is useful and acceptable. It is typically necessary to tailor interactions (information and communications) uniquely for each stakeholder.
>
> **Concerns** express the interests of stakeholders: objectives, constraints, quality and performance attributes, and other criteria, both technical and programmatic. Architectures reflect the reconciliation of concerns in negotiated, mutually consistent success criteria.
>
> **Views and Viewpoints** are the means by which an architecture is communicated to stakeholders, developers, and operators. A *view* is a tailored description that narrowly addresses in a defined way the interests of a particular *viewpoint*.

Model-based architecting provides the ability to more formally connect these notions in order to better see what views share, and thus how concerns compete. This leads to additional structural terms, as follows:

> **Models**, which comprise a substantial part of most views, are depictions or descriptions that address some aspect of the architected system or its behavior. Good models closely follow architectural concepts; good concepts are more easily modeled.
>
> **Analyses and Scenarios** are also necessary in many views. An *analysis*, frequently invoked in the context of a *scenario* (relevant circumstances or events), is any consideration of the models that draws some conclusion about them.
>
> **Trades (or Decisions or Rationales)** are the essential link between stakeholder concerns and other architectural features, explaining why an architecture is the way it is — not so much to justify concerns as to justify the choices made in response to them.
>
> **Principles (or Patterns or Guidelines)** play a prominent role in architecture. Therefore, they should be explicitly collected and justified so that the decisions they guide can be explained in a form communicable to subsequent efforts.
>
> **Elements and Relationships** are constituents of logical decomposition, conceptual and realization hierarchies, and so on that describe a system as an interacting collection of objects (abstract or not).
>
> **Properties and Functions** are inseparable from *and definitive of* elements, relationships, and the compositions that comprise them.

Model-Based Architecting Meta-models — While this list of common architecture framework notions is likely not complete, the absence of "requirements" may seem particularly surprising. Once an architectural description is complete though, requirements are merely the rendering of this description into criteria for the next layer of architectural elaboration. This is apparent in the figure below, showing framework associations among the architecting terms listed above.



Example of a structure for an architecting framework (used in Europa mission studies)

Boxes and lines represent meta-model concepts and relationships, respectively. One reads these relationships as a sentence, deriving the predicate from the line label closest to the second element, as in "A Stakeholder has one or more Concerns". Multiplicity, like "one or more", is expressed in the notation "*m…n*", which means at least *m* but no more than *n*. A single number "*m*" is equivalent to "*m…m*". "*" means any number. A missing multiplicity indicator is taken to be "1" by default.

The appropriate level of requirement detail should be no more than is necessary to describe the architecture; while associated concepts, analyses, and so on serve as an integrated, cohesive rationale. In this manner, requirements acquire important structure from framework ideas. They are seen in context, as the culmination of architectural synthesis from stakeholder concerns through concepts into validated realization that remains readily visible to stakeholders through their respective views. Instead of over-specification and slavish flow-down, an appropriately sized and connected set of requirements, driven by the

architectural imperatives for elegance and communication should be the target. Moreover, other traditional systems engineering activities, such as interface definition, V&V, and so on, can also be mapped into this structure.

Besides the associations shown in the figure, which are used to relate items in orderly, disciplined patterns, each of the terms adopted in the framework used on Europa mission studies also possesses a template, outlining the kind of information that would typically be necessary for an instance of such an item. Additional consistency rules, a predefined set of viewpoints (e.g., to capture common concerns such as mass margins, cost estimates, etc.), and associated process and methodology round out the framework.

An architecting framework can be viewed properly as a peer within a larger systems engineering framework. As long as the distinction between architecture and design is conspicuously preserved, and the principles of good architecting are maintained, architecting serves its desired outcome, where design options flow from a sound, stable architecture and are tested against this architecture, resulting in a final design that is consistent with the architecture and maintains that consistency through the challenges of implementation and operations.

A Path Forward

The development and adoption of an architecting framework (or frameworks), well integrated with current and evolving methodologies in the MBSE community, are what we aim to achieve. It is quite reasonable though to wonder whether a structured framework of simple ideas is capable of addressing the problem of architecting complex systems. There are, in fact, many ways to defeat the purpose and value of a framework, whether by constructing a poor framework, supporting a good framework with poor tools, or approaching a well-supported framework in an unappreciative or perfunctory manner. Likewise, a good framework cannot help if systems engineering outside the architecting effort takes place in the usual manner. When this is allowed to happen, "architecting" can either dissociate from reality, becoming an ivory tower exercise that serves only itself, or it can regress into a reconstruction effort, trying to assemble into some reasonable shape a plausible explanation for a point design that runs headlong before it.

What ultimately matters, therefore, is not architecting frameworks per se, but rather the elevation of architecting art and stature to a point where the purpose of a more structured approach is genuinely appreciated. A framework is merely a reflection of the architect's conviction to approach architecting in a more structured manner, informed by firmly held architecting principles. It is the *architecting* and *architecture* that deserves ultimate scrutiny, the framework being one means to that end.

Adopting a framework and embarking on a model-based architecting track is therefore not an isolated decision. Instead, this must be regarded as only part of a larger decision to address *all* of the issues of architecting outlined here. Gaining broad appreciation, preparation, and commitment for model-based architecting are essential to its success, as are team organization, management practices, training, and so on to ensure relevance and buy-in everywhere it counts. The same is true for model-based systems engineering generally. This will necessarily be a multistep process over several years, requiring the support of management willing to take this chance.

Planning a NASA Framework — On the presumption that model-based architecting is where we want to go, and therefore that a commitment has been made to the steps briefly outlined above, the natural question is how such an effort might be accommodated within the current NASA culture.

Any meaningful effort must begin with a large enough start to be relevant. Therefore, bootstrapping with a basic framework and working with it in the context of strategically selected projects is important. Targeting specific needs and actively coordinating and communicating the results would ultimately produce better frameworks with broader and more effective applicability. This must also be part of a larger, long-term strategy with a well-defined destination against which plans and progress can be judged. Included in this approach should be considerations regarding incremental development of viewpoints, a NASA-wide context for pilots, simplification of NASA procedures, automation and tool support, and allowance for transitional periods. Such concerns lead to a possible plan for NASA framework adoption, as follows:

- Establish a broadly representative NASA **focal group** to establish and refine standards, guide and invest in shared infrastructure, provide expert training and advice, and host forums for information exchange and community advancement. There are already efforts in place (e.g., NIMA described below) that could sponsor this.
- Adopt a **standard framework meta-model** from which NASA viewpoint models could be defined, each specialized to a NASA domain of interest (e.g. science, resource margins, safety, cost, launch approval, etc.).
- Consistent with the meta-model, prototype viewpoint models on substantial, strategically selected **pilot projects** (i.e., big enough and diverse enough to matter), by accommodating and refactoring information presently captured in gate products and other standard deliverables. Procedurally dictated products would nonetheless be constrained to standard format and content, but with the goal of being produced from viewpoint models with the aid of document generation tools (already available).
- Draw from pilot efforts to gradually define **official NASA viewpoint models** for one or more domains of interest, working in concert with related model-based engineering efforts. Associated templates for standard products would accompany these models, each defined in terms of viewpoint model content, but allowing simultaneous iteration of standard product definitions, where advantageous.
- Update the NASA SE Handbook and other appropriate **guiding documents** to directly cover model-based architecting. Drafts would presumably have been in iterative development as part of the pilot efforts.
- Give new projects a **product option** to produce each standard product either conventionally or through template derivation from a populated viewpoint model.
- Complete the **realignment of standard products** to best fit viewpoints models.
- Make the **final transition** to model-based architecting, where models, views, etc. are the final products.

It would be essential, of course, to wrap an assessment process around the entire effort, chartered to measure progress and decide when to move from stage to stage. Beginning to end though, this plan relies on a NASA framework for model-based architecting.

Efforts are already under way in the early stages of such an approach. Two, *among others* in NASA, are NIMA, the NASA Integrated Model-centric Architecture initiative[5], originating from the NASA Office of the Chief Engineer, and IMCE, the Integrated Model-Centric Engineering initiative[6] at JPL, which has co-sponsored the pilot effort described below. There is clearly potential for such initiatives to assume the role described above in elaborating and coordinating a model-based architecting plan, as long as they can be coordinated. With such efforts in place though, the *real* strategic decision to be made is the selection of pilot efforts. The idea is not to seek a small step that won't matter, but rather to seek a large step that can be attempted responsibly and effectively.

One Current Example within NASA — With the need to make meaningful progress in mind, the IMCE initiative at JPL has forged an alliance with study efforts for a robotic mission to Europa, a moon of Jupiter. This joint effort has been underway since 2009. A simplified framework drawn from the ideas described above and rendered in a web-based collaboration tool has been used to get everything started quickly, while development in more involved, SysML-based modeling capabilities progressed concurrently. Viewpoints are being developed as needed. The agility of a simple framework, driven mainly by model-based architecting concepts generally, rather than by an a priori set of viewpoints, has enabled the methodology to remain reasonably malleable as the study proceeded.

The collective effort has stabilized and gained acknowledgment as a viable approach; and real, deliverable artifacts have been produced — all without undue burden to the pre-project study effort.[7] Having evolved methodology in concert with delivering required products on a rigorous schedule has already shown the value of taking such strides in a real project context with real institutional backing to make it possible. This is the potent combination that makes it work and makes it worthwhile.

Still, it must be acknowledged that both mission and methodology efforts together remain very much in their formative stages. This success, so far, is due largely to a unique cooperation between visionary systems engineers and project management, supported with an adequate level of institutional support. There are several other such activities in place around NASA, but with few exceptions, these have been motivated at grass-roots levels, frequently informal, poorly funded, or generally outside sanctioned processes. The distinction of IMCE's collaboration with Europa studies has been the unusually overt intent expressed on the part of both institutional and programmatic management to actually move deliberately to a new state of systems engineering capability, methodology, and process wherein architecting is given a distinct and significant role. For model-based architecting and model-based engineering in general to take hold broadly across NASA, encouragement of such overt collaborations at the grass roots will need to continue, with strong support from NASA management.

---

[5] See https://nen.nasa.gov/web/se/nima.

[6] See Bayer, T. et al., "Update – Concept of Operations for Integrated Model- Centric Engineering at JPL", IEEE Aerospace Conference, Big Sky, Montana, March 6, 2011.

[7] See Bayer, T. et al., "Model Based Systems Engineering on the Europa Mission Concept Study", IEEE Aerospace Conference, Big Sky, Montana, March 5, 2012.

Conclusion

This is where leadership must step up to the challenge of making model-based architecting a reality. The model-based architecting approach recommended here goes beyond broadly held aspirations for model-based systems engineering generally by shaping this idea around a more structured, principled, and distinct role for architecting. This can do much to resolve present deficits in systems engineering in general, and even more importantly, for complex systems in particular. Architecture, as a product in its own right, can act as a driving force for stable, integrated technical and programmatic development. And an architecting framework, supported by model-based tools, can do much to simplify, clarify, and coordinate many aspects of NASA systems engineering practice and procedure, thereby improving the quality and performance of our missions, while helping to manage complexity within cost.

Pilot efforts alone are not sufficient. What remains to be added is the critical mass of support needed from leadership at the center and agency levels. The purpose of this report has been to try to explain why model-based architecting makes sense for NASA, and therefore why such plans and commitments are warranted.

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.