CESEM

Center for Earth Systems Engineering and Management

My Kingdom for a Toolsmith! The IRIDIUM system specification through the lens of Actor-Network Theory

Tom Roberts

ASU-SSEBE-CESEM-2010-RPR-001
Research Project Report Series

December 2010

My Kingdom for a Toolsmith!
The IRIDIUM system specification through the lens of Actor-Network Theory

Tom Roberts
December 6, 2010

Abstract

Most would agree that telecommunications systems are socially constructed. Since "communication" tends to involve people, it seems obvious that people should impact the creation of such systems. But it is far less obvious that the *specifications* for such systems should be noted for their social construction. As marvelous and technical as the system is, we must not forget the important technological artifact known as the specification that came before it. This paper tells the story of the social construction of the IRIDIUM system specification as viewed through the eyes of a popular socio-technical systems (STS) analysis tool. Actor-Network Theory (ANT) is employed to elucidate the culture of the Motorola requirements engineering process while describing some of the primary actors and their lively interactions as they strove diligently to produce the "perfect" specification. Throughout, it will become obvious that just as the kingdom was lost "for want of a nail," so the IRIDIUM system specification was *nearly* lost for want of a toolsmith.

Introduction

The 1990s witnessed an unprecedented event in both the Space and Telecommunications technology sectors: Motorola developed the world's first (and only) global, space-based telephony system boasting service anytime, anywhere on the planet. The system was dubbed IRIDIUM™ because the initial plan called for 77 satellites in low-earth orbit and 77 is the atomic number of the element Iridium. Prior to this announcement, cellular telephony systems in the U.S. were beginning to expand but still offered limited (mostly urban) coverage, and the idea of "anytime, anywhere" was compelling. IRIDIUM was to fill an important niche in this world of telecommunications.

At the announcement, it was immediately clear to large systems engineering firms (e.g., TASC, SAIC, Booz-Allen-Hamilton), who were regularly employed in large government programs that this would be a target rich environment for their wares: system engineering services. This was because those of that ilk knew that Motorola had limited experience specifying, designing, building, deploying, operating, and maintaining a system of this magnitude. At the time, Motorola's "comparative advantage" (*sensu* Ricardo, 1817) was in designing and developing small devices: radios, phones, pagers, and microprocessors. For their own government jobs, they would sometimes deploy communications "systems" to outfit a large van or armored personnel carrier, but most Motorola systems remained small. They simply didn't know how to engineer a complicated system like IRIDIUM. So it was, then, that Motorola added dozens of system engineering service providers to the IRIDIUM system team—which already boasted big players like Raytheon, Lockheed-Martin, and McDonnell Douglas—each contributing their experience, and each hiring as fast as it could to meet the demand of this huge development effort.

During the mid-1990s, I supervised a staff of over 40 engineers working on the IRIDIUM system. Each engineer performed analyses and research, and developed specification and requirements documentation using a Motorola-mandated model-based system engineering (MBSE) tool known as RDD-100 (a product of Ascent Logic Corporation at the time). This important mandate was the "make or break" for a system engineer on the IRIDIUM program. Working with the cumbersome RDD-100 package was complicated, frustrating, and *absolutely required*. This led to the rise of a new breed of system engineer who quickly mastered the quirks of RDD: the toolsmith. With so many engineers

required to use so difficult a tool, the toolsmith became the MVP of the specification development effort. But before I get too far into the story, I must fill in a bit of background.

Systems Engineering 101

As a fresh-out, recently graduated engineer, still wet behind the ears, I walked into the first day of my new job full of anticipation, eager to embark on a lifetime of learning. I was met by my supervisor who guided me into a lab and introduced me to the senior engineer who would be my mentor. After Byron said "Hi" and "Welcome aboard," he walked over to a collection of equipment racks, affectionately patted one, and said "Well Tom, I guess the first thing you need to know is that the PE arms the RIU DMA every MFS." I very politely interrupted and said, "Begging your pardon, I don't think that's the *first* thing I need to know."

Obviously, that lesson has stuck with me for over 25 years. I would like to say I never did that to a junior engineer, but it is probably not true. In fact, we have all been put in situations where we have found it necessary to, as the saying goes, "sink or swim." To a certain extent, a reader who is unfamiliar with systems engineering principles might be lost in the sea of terminology and "specification practices" that are outlined herein. In the interest of ensuring that you are doing more swimming than sinking, this section will highlight some of the terminology and practices of a system engineer. While this by no means constitutes a complete explanation of the discipline of systems engineering, this introduction to the concepts will provide you enough background to understand what comes later, and will also allow me to introduce the IRIDIUM system by way of example.

Figure 1. IRIDIUM System Overview Showing Primary Subsystems and Interfaces
(Source: IRIDIUM System marketing brochure, c. 1993. Illustration by Dale Glasgow)

Jim Helm, Senior Systems Engineer in Motorola's Satellite Communications Division, and manager of the system engineering effort, described the purpose of IRIDIUM as follows:

> The IRIDIUM System will provide a worldwide communications service for voice and subscriber data between any two IRIDIUM Subscriber Units (ISUs) as well as between any IRIDIUM Subscriber Unit and a local [public switched telephone network] customer. It will also provide paging for its customers (Helm, 1997, p. 1).

This is aptly depicted in Figure 1 which shows an artist's conception of the IRIDIUM system. As the first entry in the system engineering lexicon, understand that the "system" is the highest level representation of what is being built. Generally, a large system will be comprised of subsystems. It should be noted that subsystems are themselves systems and *system decomposition* can continue down to an arbitrarily small terminal system. Sometimes smaller subsystems are called other names, for example, IRIDIUM referred to them as "segments". The first goal of systems engineering is to completely decompose the system into its component parts, the *subsystems*, and to identify the *interfaces* between them. The $5B IRIDIUM system was comprised of four primary subsystems: the Space Segment which contained the satellite vehicles (and also included the satellite launch vehicle interfaces), the System Control Segment that managed the satellite network while in orbit, the Gateway that interfaced IRIDIUM to the terrestrial telephone systems known as the Public Switched Telephone Network (PSTN), and the IRIDIUM Subscriber Unit which was the IRIDIUM phone.

For our purposes, it matters very little what each segment contributed to the overall system. What *is* important to us is that once a system has been decomposed into its major subsystems, *subsystem engineers* (SSEs) are assigned to be the specialists for the functions that occur within their respective subsystems. They are also tasked with ensuring the *interfaces* between their subsystem and the other subsystems are properly specified and in this effort are assisted by the system engineers. IRIDIUM *system engineers* (SEs) maintained purview of the entire system, that is, they understood the system at the highest level and were responsible for ensuring the system would function as intended once it was built.

To ensure the system was built properly and functioned adequately, the system engineers used a *specification*. A system specification is a detailed list of all the requirements for the system. Very simply, it's a definition of the function and performance of a system. Each of the system *requirements* must be *allocated* to a *subsystem* and assigned to a subsystem engineer. The goal for a specification is that it be "perfect" as defined by four keywords: complete (no required function is missing), consistent (no internal conflicts), validated (the requirements are real and necessary), and verifiable (can be tested and proven to be satisfied). For example, if someone suggested a requirement like "IRIDIUM shall provide personal communications anytime, anywhere," it becomes the job of a system engineer to decide what that means and what parts of the system are involved in satisfying that requirement. This process of requirement analysis and allocation to subsystems is iterative and collaborative, leading to the myriad small functions and features of a system like IRIDIUM.

To assist them in their jobs, system engineers use a variety of Computer-aided Design (CAD) and Computer-aided System (or Software) Engineering (CASE) tools. One such tool was RDD-100, which implemented a method known as Requirements Driven Development to ensure a "perfect" specification in all the ways mentioned above. Usually referred to simply as RDD, this tool was very powerful, capable, thorough, and complete, but it was also extremely complicated and difficult to master. Even after extensive training, the tool could require months to use effectively.

In summary, *system engineers* who have purview of the entire *system* first *decompose* the system into *subsystems* for which *subsystem engineers* take ownership. *Interfaces* between subsystems are identified and thereafter all system *requirements* are *allocated* to subsystems and interfaces in a process of negotiation that will lead to appropriate functional and performance requirements. The goal is a *perfect specification*: complete, consistent, valid and verifiable.

Once the need for a system is established and the funding has been secured, it seems a simple next step to write a specification and commence building the system. For a system of the immensity of IRIDIUM, however, this is not a simple challenge. Due to their proprietary nature, specific costs were never published, so it is somewhat difficult to provide an idea of the size of the effort. Based on my

extended tenure and broad exposure within the effort, however, I estimate the specification development effort *alone* involved hundreds of engineers, full-time, for several years. My conservative estimate suggests that developing the specification for the IRIDIUM system cost over $100M. This kind of money forces management to take the effort very seriously. Results were vital.

Actor-Network Theory and the IRIDIUM Specification

My selection of ANT to analyze the development of the IRIDIUM system specification was based on its similarities with software development and business process engineering methods with which I am familiar (cf., Jacobson, 1992; Jacobson, 1995). Such methods are used regularly in the *creation* of specifications, so the application of ANT to analyze the social construction of the specification artifact seemed a natural next step.

Though it is not likely ANT has been employed in the analysis of a specification development effort (a literature search unearthed nothing similar), ANT's utility is not foreign to the information systems domain, and it has been successfully applied to a variety of other projects involving information technology. For example, ANT has been used to analyze systems in the health care sector (Cho et al., 2007) and Enterprise Resource Planning systems (Elbanna, 2008). Even the automated baggage handling system at the Denver International Airport was evaluated using ANT (Mahring et al., 2004). In fact, Walsham suggests "no particular context or information system type can be excluded as a possible application area for the theory" (Walsham, 1997, p. 477). Further, it is clear that artifacts like a specification *should* be considered in consort with the network of actors and artifacts surrounding them. Law points out "the stability and form of artifacts should be seen as a function of the interaction of heterogeneous elements as these are shaped and assimilated into a network" (Law, 1987, p. 113), making ANT a logical selection.

Callon's presentation of the primary features of Actor-Network Theory (Callon, 1986) provides a solid framework for analyzing the development of the IRIDIUM system specification. In the analysis below I will rigorously follow his three principles. First, *generalized agnosticism* will be demonstrated through my impartiality toward both scientific and social dimensions of the specification development effort. No point of view is privileged. Second, *generalized symmetry* is observed as I employ a single repertoire for both social and technical conversations. Finally, *free association* forces me to treat all actors equally, whether they are natural or social, human or mechanical.

I will employ Callon's *four moments of translation* (Callon, 1986) in a somewhat linear manner for purposes of clarity (though linearity is neither implied nor expressly required in Callon's work). Treating them as "stages" herein will allow me to explore each step in detail as it relates to the IRIDIUM system specification. Callon's final "step," dissidence, also turns out to be very important as we approach closure for the artifacts under analysis.

Problematization

Callon's first moment of translation involves inter-definition of the actors (Callon, 1986). That the actors are *inter*-defined is an important feature that becomes more apparent in the interessement, but the problematization reveals one of the more powerful features of ANT: the actors need not be human. While ANT works effectively with human actors, it also enables considerable depth of analysis for non-human actors—as long as they are given appropriate voice. The trick is to define them in such a way that there is a hub where they all come together at an obligatory passage point (OPP).

Due to the immensity of the IRIDIUM specification project, I'll be forced to limit my remarks to a mere handful of the actors involved; three of which are human, two non-human. While you've not yet been introduced to the key player (the toolsmith), you have already been exposed to four of the actors in passing in the introduction to system engineering principles: the system engineer, subsystem engineer, specification, and RDD-100. Two other actors will appear in the discussion of dissidence (ConOps and

Flowtool), but these will be further defined at that time. The actors in the analysis are described in the paragraphs that follow.

Management declared to all, "Thou shalt use RDD and make a perfect specification!" Recall that RDD was a CASE tool that was incredibly powerful, yet extraordinarily difficult to master. This decree struck fear in the hearts of the system engineering team because IRIDIUM was going to be hard enough itself without adding the complexity of a new and sophisticated tool. These were engineers who understood important details about the entire system so they could effectively orchestrate the competing requirements and allocate them to appropriate subsystems. Most system engineers were senior and experienced in large systems integration, but had no exposure to tools. Immediately individual SEs started saying, "I have a system to design. I don't have time to learn RDD!" And, lamenting the quirks of the new specification tool, they'd wonder, "How can I write *that* requirement in RDD?" Once some alacrity with the tool was gained, it seemed there was always some roadblock that would engender remarks like "RDD won't like that!" And eventually, once enculturation was complete, engineers would start instructing others with, "You're supposed to do that in RDD!" Amidst all this, the system engineers were reminded that their role was limited to *system level* specification. They were not to force specific implementation details on the subsystems. Instead, they were to communicate only *what* the system must do and flow the requirements down to the subsystem engineers who would determine *how* it would be accomplished.

In the meantime, the subsystem teams were given high-level budget figures and were directed to start conceptualizing what their respective parts of the system must do to deliver on the system-level requirements. The subsystem engineers were individuals who focused their expertise on the functions and performance of their specific segment and were qualified to contribute to the definition of the system level interfaces. SSEs were instructed to cooperate with the SEs, but knew in the back of their minds that budgets were becoming less flexible and schedules were being pushed. On many occasions they could be heard to say to the SEs, "Skip your RDD modeling. Just tell us what you *need*!" Or, when presented with a robust RDD model that portrayed important requirements but in RDD's inscrutable manner, "What do I do with *this*?" Eventually, as the schedule continued to burn, it was much more frequent for the SSEs to simply refuse to take new requirements with "Stick a fork in me, I'm done!" which was usually followed quickly by "Trust us! We know what we're doing!"

Giving the specification a voice in the problematization is an important feature of ANT. The specification was a principal actor in that it was the end product of the effort. The specification was a demanding taskmaster, always wanting to be perfect. It knew it must pass muster under the scrutiny of the managers and engineers, so many times it demanded to be re-written to meet its own strict requirements. There were also many computerized tests that the specification must pass to ensure it was formal and complete, so frequently it would remind its authors to ensure each requirement was really verifiable, or that each had a "shall" statement that could be satisfied. The specification was also aware of the fact that it would eventually become the real system, so stray requirements had to be finalized and allocated. In this regard, the specification considered RDD-100 its friend—because RDD could ensure the specification was perfect.

RDD-100 was the CASE tool selected to assist engineers in specifying the system. It was capable of formally representing the most complicated of highly-interactive systems. Though it had some noteworthy quirks, it was certainly up to the task. This tool could not only model the functional requirements of the system, but it could also assist in defining the performance requirements, supporting engineers through the entire system development effort and into integration and test (Roberts & Farley, 1998 provides a reasonable overview of RDD's utility over the entire system lifecycle). The most difficult feature was its "cost of entry." RDD "contributed" to the specification development effort by being hard to use and hard to get along with. While an uncompromising master, it *was*, in fact, capable of generating the perfect specification. But it had an attitude. More frequently than not it stood in the way while the engineers were reminded that it must be used. RDD frequently reminded engineers they were "doing it wrong" or that they'd left something out, resulting in an incomplete specification.

If RDD was so hard to master, it's valid to wonder why management would force its use. The reality is that it was hard to use by the staid masters of the system engineering art. Their reluctance to adopt the tool was largely because they'd "never done it that way before." RDD's power was unquestioned. The pundits of the day were saying:

> The object-oriented approach is ideal for system decomposition into subsystems—the basis for systems-level analysis. The RDD-100 system includes an object-oriented database that encapsulates complex system information. You can descend through multiple levels of detail until you arrive at a component as basic as a resistor (Brown, 1995).

Using RDD was, in fact, a *good* decision, but, given the culture, it might not have been the *right* decision. It is arguable that for the team developing IRIDIUM, "old school" might have been a less stressful approach. But, the decision was made and the team was faced with a dilemma: How could they make it work?

This is where the toolsmith stepped in and provided a solution that was unrivaled. These were among a younger generation of system engineers able to master RDD more quickly. Further, there were specifically skilled engineers that could make RDD sing and dance in short order. Paraphrasing Langdon Winner, these were engineers whose skill with tools "became woven into the texture of their everyday existence; the devices, techniques, and systems had shed their tool-like qualities to become part of their very humanity" (Winner, 1986, p. 12). These were known as toolsmiths. Usually, a toolsmith is a software practitioner that specializes in software tool use and feature exploitation. Sometimes this is done through a programmable interface; sometimes it can be accomplished with scripting languages that are programmed to operate the tool like a software robot. In the case of IRIDIUM, the toolsmith mastered a short list of tools and RDD was at the top of the list. Thankfully, when the dilemma reached crisis proportions, the toolsmiths raised their hands and said, "Tell me what you want, I'll drive the tool."

Obligatory Passage Point (OPP)

If the system engineers hoped to be successful in creating a worthy system, if the subsystem engineers held onto any visions of being able to finish on time and on budget, if the specification maintained its dreams of perfection and purity, and if RDD had any plans of being able to overcome its antisocial behavior and actually be used on this project, then there must be a toolsmith. IRIDIUM needed the toolsmith—someone who would stand in the gap and bring all the groups together to generate a perfect specification. In this manner, the toolsmith became the ANT obligatory passage point (OPP) as depicted in Figure 2.

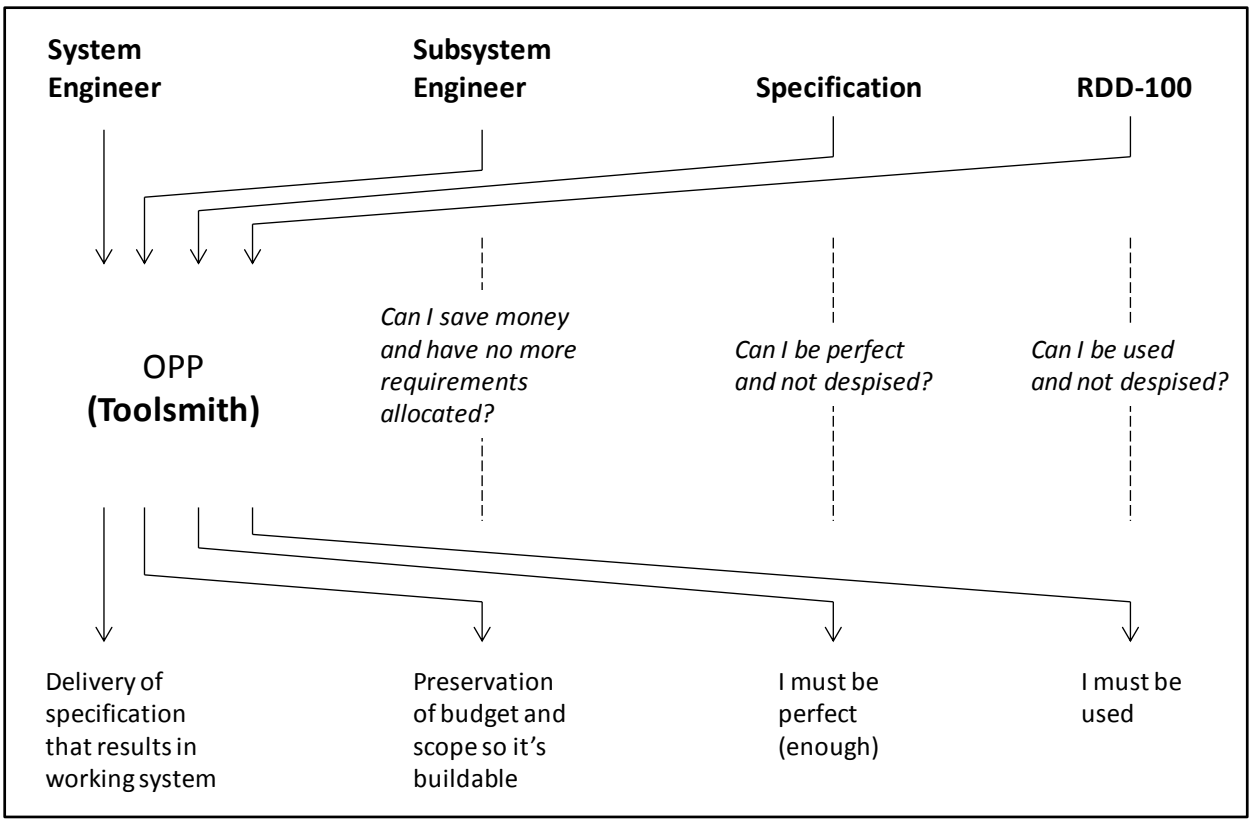

Figure 2. ANT *Problematization* showing Obligatory Passage Point

Interessement

Interessement deals with the "interest" of each actor. Callon suggests it is how the actors are locked into their roles (Callon, 1986). During this moment of translation we learn what draws the actors together in a profit-sharing relationship. This step defines the mutual benefit they derive, the win-win resolution.

The system engineers (SEs) are responsible for the overall specification, but have no direct authority to force subsystem engineers (SSEs) to contribute well. SEs can only "suggest" and attempt to maintain a feel-good, influential approach to cooperation on the specification team. Their problem is that they also must fulfill their duty to release a "perfect" specification containing all the salient details that only the subsystem engineers can provide. SEs, then, become indebted to the toolsmith for off-loading the tool responsibilities so they can manage the personal interface with managers and subsystem engineers with whom they must collaborate. Further, the SEs can pour all their hopes and dreams for the system onto the toolsmith and trust that they are faithfully rendered into proper RDD models and hence reflected in the specification that is ultimately generated.

Subsystem engineers must also contribute their part to the specification effort using the RDD-100 tool to analyze and provide inputs. Their problem is they have no time for training, and little support from their management for such time away from work (even though training is provided). In this regard, the toolsmith who can take raw information and enter it into RDD is a godsend to the subsystem engineers. If subsystem engineers can "deliver" on their part, they will, in turn, be loved by the system engineers who are desperately trying to release a perfect specification. In this manner, the toolsmith "weakens the links" (Callon, 1985, p. 9) between the SE and SSE and becomes the solution to the problem.

But interessement doesn't end there. The specification loves the toolsmith because otherwise it must languish in imperfection, struggle for well-being, and be relegated to obscurity. Though the management mandate made it clear that the specification must exist, there was no guarantee its existence

would be high quality. The toolsmith made it possible for the specification to excel toward perfection in a way it could never have done otherwise.

RDD also loves the toolsmith. Like a self-centered pet cat, RDD has its back scratched in all the right places while continuing to heap "attitude" on the toolsmith. The toolsmith knows all the right buttons to push and all the right features to exploit. He keeps RDD happy and fulfilled. And still, the toolsmith finds his role fulfilling as well. Not only does he get to make everyone happy (SE, SSE, managers, etc), he gets to learn sophisticated CASE tools, optimize their throughput, participate in creating the biggest system in the world, and learn a bit about system engineering. Plus he gets to write software while having some fun with tool automation. In these ways he makes himself indispensible, the centerpiece of the team, as shown in Figure 3.

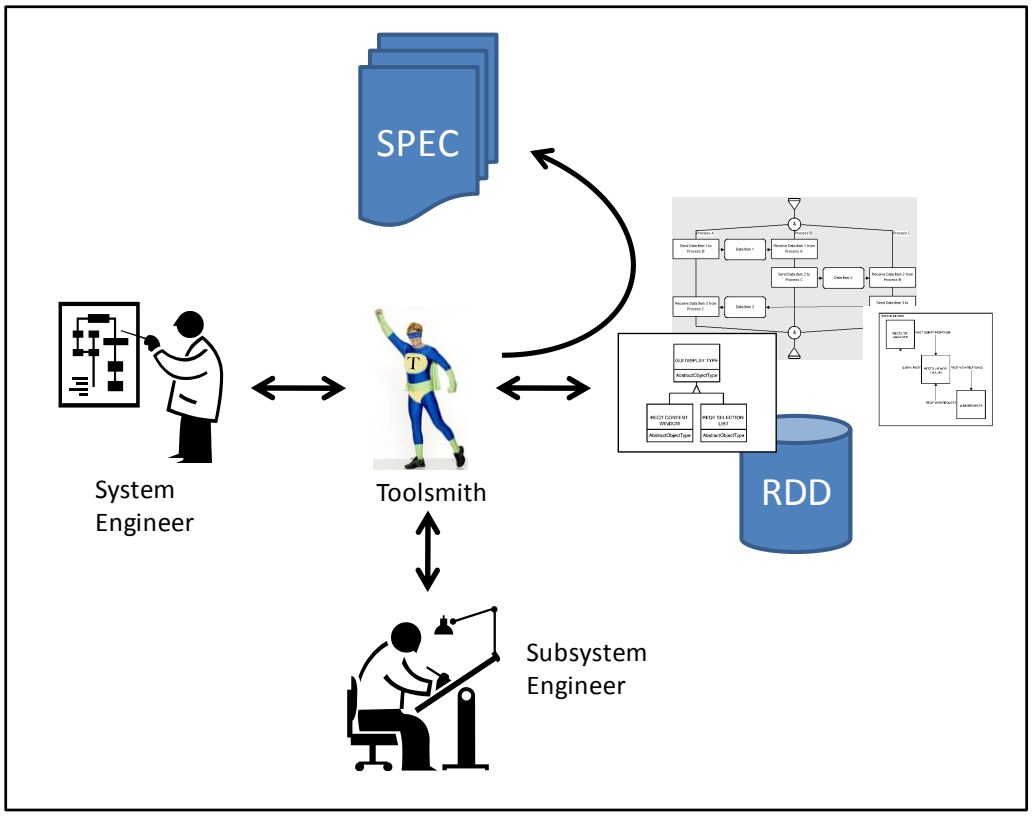

Figure 3. *Interessement* showing the Toolsmith as the Centerpiece

Enrolment

Callon's enrolment translation consists of an ongoing negotiation and engagement process that leads to success (Callon, 1986). The manner in which the toolsmith accomplished this in the IRIDIUM specification effort can be best explained as an implementation of the Kano methodology as adapted for the business model known as *The Phoenix Imperative* (Roberts, 1997/2010; Boar, 1993).

The toolsmith focused the actor-network by taking the central role. As shown in Figure 4, a Kano threshold was established and the network coalesced around the toolsmith who met all the threshold qualifications. Thereafter, by performing regularly in keeping the effort moving forward, the toolsmith retained the engagement of all the actors. This constituted a period of linear return on investment (ROI) where any tasking assigned to the toolsmith could be expected to bring quality results. Occasionally, the toolsmith would deliver a Kano excitement driver and attract everyone's attention by doing something previously impossible (usually through scripting or tool automation). Such excitement drivers had exponential ROI and would re-engage any actors whose interest may have flagged. Eventually, the truly

clever toolsmith repositioned these advanced capabilities as entry-level threshold attributes effectively locking out the competition and perpetuating his role. It was in this manner that the IRIDIUM specification development effort made great strides. The toolsmith was able to optimize everyone's (and everything's) performance.

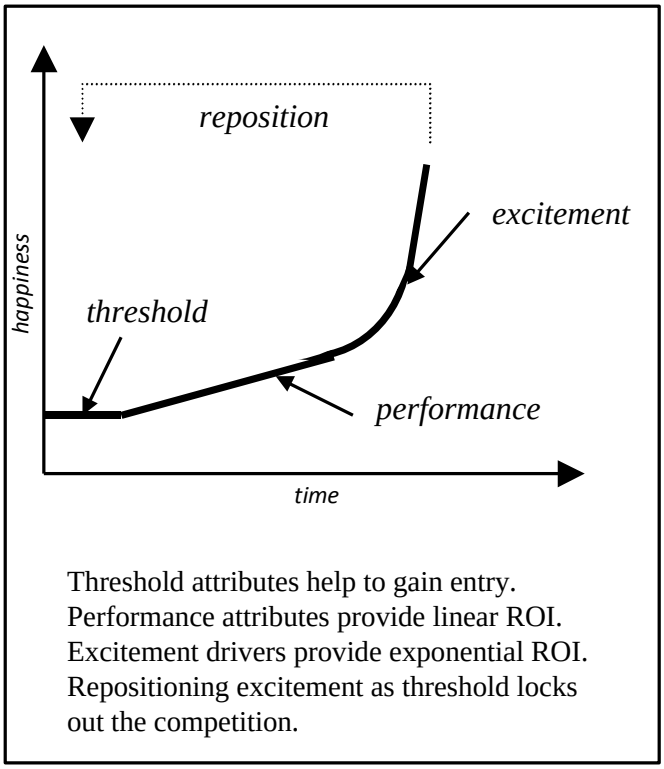

Figure 4. "Enrolment" ala the Kano Method

Mobilisation

Andrade interprets Callon's mobilisation translation as "the stage when actants become spokespersons representing the network" (Andrade, 2010, p. 363). While this is important, it is vital to understand that mobilisation demands *appropriate* representation. Mahring adds the flavor of *loyalty* in representation when he describes mobilisation as involving "use of a set of methods to ensure that allied spokespersons act according to the agreement and do not betray the initiators' interests" (Mahring et al., 2004, p. 214).

The interessement and enrolment instigated by the toolsmith was so strong for the IRIDIUM specification effort that to the surprise and delight of all, *the specification started speaking for the teams*. This marked a measure of maturity toward which specification development teams strive, but only sometimes achieve. It was not uncommon to hear conversations containing phrases like "What's the spec say?" or "I'll have to check the spec." Highest honor is accorded the specification when it is deemed "right" in lieu of a particular implementation, such as, "well, I don't know how you did it, but the spec is 'right' so it needs to match the spec."

As the specification continued to mature, and as engineers began to trust it, more requests came to the toolsmith for exported content, or even for training in how to locate and export the content themselves. And so it was that the specification faithfully represented the team to the outside world. But this successful mobilisation was all too brief. It wasn't long until the dissidence set in.

Dissidence

During Callon's period of betrayal and controversy, the interessement and enrolment can often be seen to fail. Sometimes the issues reach all the way back and call into question the problematization, indicating, perhaps, that the actor-network was never established properly by the analyst. In these latter cases, it is important to learn from this that it isn't what actually happened that was "wrong" per se. Only that the analyst failed to appropriately fit the reality into the model. More frequently, however, what is witnessed is that the actor-network operates as expected for awhile and then dissolves, either due to betrayal, or as a logical consequence of elapsed time (i.e., all projects end, but not all fail, in both cases, however, the team dissolves). This is what was witnessed in the IRIDIUM specification effort.

On times scales as long as the IRIDIUM specification development effort (several years), there are many instances of dissidence. For our purposes, there are two important actors that appear later in the game and are important to the discussion of dissidence. These actors were an alternative "CASE tool" known as Flowtool, and an alternative "specification" document known as a Concept of Operations, or, ConOps. It is noteworthy that I have conspicuously employed quotation marks around "CASE tool" and "specification" in the previous sentence. This is intended to imply that Flowtool was *not* a CASE tool in the sense that RDD was, and *neither* was the ConOps a specification in the same sense the real specification was. Importantly, they were not charlatans masquerading as more sophisticated tools, they were simply late entries that filled important roles and satisfied particular needs.

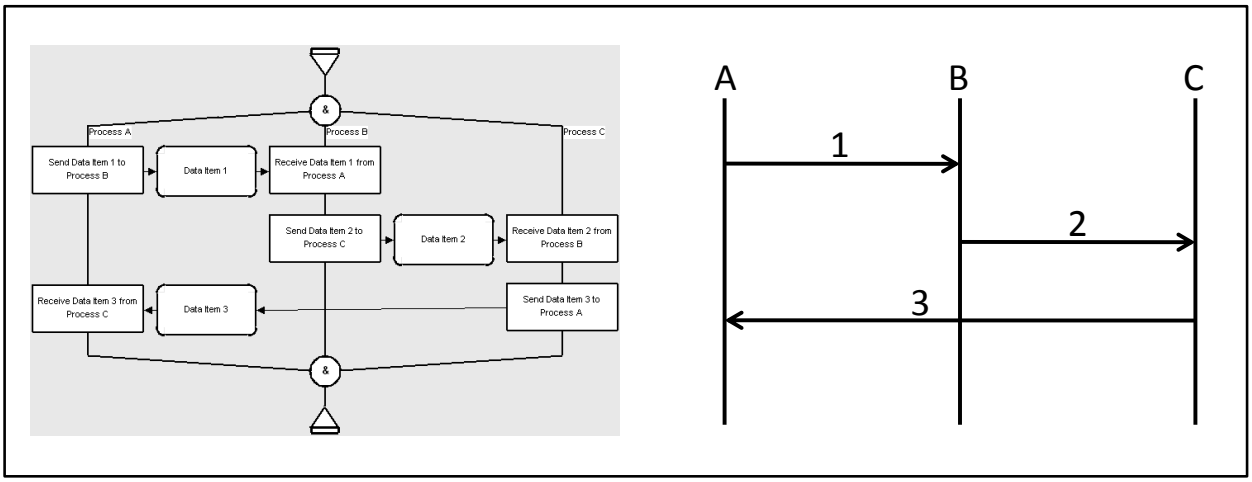

Figure 5. RDD Behavior Diagram (left) and "Identical" Flowtool Representation (right)

Flowtool was a homegrown (developed internally by a Motorola contractor), text-based language (picture a very simple software programming language) from which message sequence diagrams (see Figure 5) could be generated as a product of compilation. This tool was anything but rigorous and was incapable of adequately modeling a system of any significance. Its primary benefit was that the output product was easily interpreted by just about anyone after a few minutes of instruction—and it "mostly worked" for the goals of the subsystem engineers. As shown in Figure 5, not only does the Flowtool representation appear cleaner, it is more quickly assimilated, and far easier to draw. It should be obvious why such a pictorial representations soon were in general use and created dissidence within the specification teams who were required to use RDD to make the equivalent figures. Almost as a testament to their ongoing productivity and resilience, the toolsmiths were able to quickly develop software that would convert one visual representation into the other with very little effort.

Dissidence also came in the form of smallish documents known as Concepts of Operations, or ConOps. These were informal, English language, prose documents that described with a broad brush how a part of the system would work. Such documents are widely used as simple communications tools and certainly have a place in system development, but they are insufficiently rigorous and far too informal to

be the only source of requirements for a system. There are fundamental and important differences between a specification and concept of operations, not the least of which is that a ConOps contains only minimal performance requirements and still less requirement allocation and traceability. Still, engineers started writing these simple documents and they began to gain a following. Senior engineers, who were too proud to learn RDD or work with a toolsmith, were garnering RDD "exemptions" and plunging headlong into the creation of operations concept documents, preferring the free-form, stream-of-consciousness approach that communicated well to those not indoctrinated into the complexities of RDD.

In addition to these, there was an RDD bottleneck that came from too many requirements and too little time. This occurred primarily due to lack of planning on the part of management inexperienced with RDD and the employment of such tools. Management also granted "overrides" that allowed subsystem engineers to "figure it out" between themselves (without the involvement of system engineers) and with only a promise that such "hallway baselines" would be recorded in the specification. Further there was a tendency to suggest that difficult issues become "integration and test requirements" that could be figured out as the subsystems were joined together into the larger system.

In a nutshell, specification users started fleeing, finding the occasional ConOps to be more interesting reading. Add to this the rumors that RDD was going to be replaced by some other tool and the betrayal was complete. Once the mandate was called into question, RDD could no longer continue to demand it be used. The specification had lost its voice. The system engineering teams had lost their purpose. Or, had they? It was around this time that everyone looked up and realized the IRIDIUM system was built, was being integrated, and was starting to deliver on its promised function.

Closure

Closure was seen in early and extended success for the problematization. The toolsmith *really did save the day*, drawing the team together and providing a mechanism for success. Interessement and enrolment were also quite successful. The toolsmith was able to greatly extend the duration of use for the RDD CASE tool and the period during which formal specification practices were employed by the Motorola team. Mobilisation happened, but it was fragile and temporary. Still, it lasted long enough to be observable and effective. The specification spoke, but it wasn't long before dissidence set in and its voice was lost in the sea of dissent. Fortunately, the IRIDIUM system itself was reaching closure, and the role of the specification was diminishing anyway.

Summary

Summarizing the ANT analysis reveals that all goals were satisfied. Generalized agnosticism was demonstrated in that all actors had hopes, dreams, roles, dilemmas, victories, setbacks, and final dispositions. Generalized symmetry was demonstrated as actors were given equal voice with equal weight. Free association was demonstrated as actors were chosen from the human and non-human realms. Callon's four moments of translation were demonstrated and proven to be effective mechanisms for analysis.

Dissidence was obvious though it did not specifically refute the "problematization." Instead, it occurred at the relational "end of life" for the "interessement." As the specification did its work of defining the system and the system was "built to spec" the system engineers moved into new roles and the specification teams were no longer compelling relationships to maintain.

Conclusion and Outcomes

While it is clear that the IRIDIUM specification was socially constructed it is also clear that the specification development effort and the processes and tools associated with it did a significant degree of shaping the social dimension as well. Sheila Jasonoff makes this point strongly in defining co-production: "the workings of science and technology cease to be a thing apart from other forms of social activity, but

are integrated instead as indispensible elements in the process of societal evolution.... the realities of human experience emerge as the joint achievements of scientific, technical and social enterprise: science and society, in a word, are co-produced, each underwriting the other's existence" (Jasonoff, 2003, p. 17). Social construction is rarely a one-way street.

In this analysis, both ANT and SCOT have shown their strengths. ANT has demonstrated its power across the board from problematization through dissidence. It adds significant value in giving artifacts a voice. ANT also highlights how the important relationships forged during interessement and enrolment can be vital in the later stages of mobilisation and dissidence where trust between actors is paramount. SCOT, in its turn, has demonstrated the importance of the evolution of artifacts. Artifacts do evolve over their lifecycle. This can be obscured by the ANT approach which would tend instead to model evolution through a change in voice or representation.

Where are they now? The specification continued to shout to deaf ears and was last heard saying: "Don't forget to leverage me for IRIDIUM-NEXT!" RDD was sidelined because of its pride. It was avoided due to its personality flaws and hubris led to its downfall. It did receive some minimal follow-on work on Teledesic, but its reign of terror ended with the demise of that system. The system engineers became the system test team. That transition was fostered by the thought that "You specified IRIDIUM, now make it work!" and it was more of a punitive sentence. The subsystem engineers reverted to development work within the subsystem teams or moved to the test team. Essentially they were assimilated by the IRIDIUM development army—which was massive.

And what happened to the toolsmiths? They prospered. They were useful to the end and went on to serve in a variety of roles on many other programs and with many other tools.

No. IRIDIUM was *not* lost for want of a toolsmith!

Future Work

Future efforts on the analysis of IRIDIUM system development would bear fruit if focused in four primary areas: (1) the inclusion of additional actors in the analysis, (2) employment of the latest growth and expansion of actor-network theory itself, (3) exploration of other socio-technical themes, and (4) use of other analysis tools and approaches.

First, additional actors could be analyzed to provide for a richer set of interactions. No effort was made herein to analyze the role, for example, of contractors v. employees of Motorola—and there were *significant* issues to overcome in this area (see Roberts, 1997/2010). Managers should be included in order to explore the directive and mitigating influences they made. The role of the tool vendor was also important. A specification review board held considerable sway in defining allowable levels of perfection. Finally, including non-human actors like budget and schedule would be illuminating.

Second, in his later work, Callon appears to have augmented ANT to include the concept of devices. There are three kinds of devices: substantial (what flows around the actor-network), material (the substance of the substance—if you will—that flows, that is, what it is made of, how it is shaped), and procedural (what instigates and promulgates the flow). The inter-relationships of these devices establish the network (Callon, 1991; Callon, 1999). These concepts are worthy of pursuit in the IRIDIUM space. For example, for the IRIDIUM specification effort, the *substantial* device might be "system knowledge," the *material* device is the specification itself, that is, system knowledge codified in the RDD tool, and the *procedural* device represents the many interactions between engineers, the toolsmiths, and the RDD tool. This analysis could be fruitful and might actually instruct the industry on potential improvements to the process of specification development.

Third, other socio-technical themes such as the politics of the specification, the tools, and the practices should be analyzed. The artifacts (e.g., Flowtool charts) gained political power of their own and exerted influence in the daily machinations of the development effort. Technological optimism and momentum could be evaluated to determine the extent to which they carried and thwarted the team. Women's themes should be explored. Of the several hundred system engineers in the Motorola Satellite Communications (SATCOM) division, only a few percent were women. To not comment briefly on the

feminist agenda at work within SATCOM seems negligent, but in the interest of space and focus, these must be relegated to future research.

Finally, the specification development effort should be analyzed from the standpoint of Jasonoff's co-production. There is no question the "shaping" was bi-directional.

References

Andrade, A. & Urquhart, C. (2010). The affordances of actor network theory in ICT for development research. *Information Technology & People*. 23(4) 352-374.

Boar, B., (1993). *The Art of Strategic Planning for Information Technology*, John Wiley & Sons, Inc. pp. 210-211.

Brown, C. (1995, September). Détente comes to systems engineering. *Electronic Engineering Times*. p. 40. Retrieved December 1, 2010, from ABI/INFORM Trade & Industry. (Document ID: 10803439).

Callon, M. (1986). Some elements of a sociology of translation. In Law, J. (Ed.) *Power, Action and Belief*. London: Routledge. Pp. 196-223.

Callon, M. (1991). Techno-economic networks and irreversibility. In Law, J. (Ed.), *A Sociology of Monsters: Essays on Power, Technology and Domination*. London: Routledge. Pp 132–161.

Callon, M. (1999). Actor-network theory – the market test. In Law, J. (Ed.), *Actor Network Theory and After*. Oxford: Blackwell.

Helm, J. (1997). IRIDIUM Worldwide Personal Communication System. *AIP Conference Proceedings*. 387(1) 979-984.

Jacobson, I. (1992). *Object-Oriented Software Engineering – A Use Case Driven Approach*. Reading, MA: Addison-Wesley and ACM Press.

Jacobson, I. (1995). *The Object Advantage – Business Process Reengineering with Object Technology*. Reading, MA: Addison-Wesley and ACM Press.

Jasanoff, S. (Ed.) (2003). Ordering Knowledge, Ordering Society. *States of Knowledge: The Co-Production of Science and Social Order*. New York: Routledge.

Law, J. (1987). Technology and Heterogeneous Engineering: The Case of Portuguese Expansion. In Bijker, W., Hughes, T. & Pinch, T. (Eds.) *The Social Construction of Technological Systems*. Cambridge, MA: MIT Press. Pp. 111-134.

Mahring, M., Holmstrom, J., Keil, M. & Montealegre, R. (2004). Trojan actor-networks and swift translations – bringing actor-network theory to IT project escalation studies. *Information Technology & People*. 17(2) 210–238.

Pinch, T. & Bijker, W. (1987). The Social Construction of Facts and Artifacts. In Bijker, W., Hughes, T. & Pinch, T. (Eds.) *The Social Construction of Technological Systems*. Cambridge, MA: MIT Press. Pp. 17-50.

Ricardo, D. (1817). *On the Principles of Political Economy and Taxation*. London: John Murray.

Roberts, T. (1997/2010). *The Phoenix Imperative: An Alternative Maturity Model for Systems Engineering Service Providers*. Available at: http://hdl.handle.net/2286/R.A.106179.

Roberts, T. & Farley B. (1998). Regenerative Verification and Validation: Moving Forward from Reverse Engineering. *Proceedings of the 6th International Conference on Reengineering*. IEEE Press.

Smith, S., Rose, M., & Hamilton, E. (2010). The story of a university knowledge exchange actor-network told through the sociology of translation. *International Journal of Entrepreneurial Behaviour & Research*. 16(6) 502-516.

Walsham, G. (1997). Actor-network theory and IS research: Current status and future prospects. In *Information Systems and Qualitative Research*, Lee, A., Liebenau, J., & DeGross, J. (Eds.). London: Chapman and Hall. Pp. 466–480.

Winner, L. (1986). *The Whale and the Reactor*. Chicago: University of Chicago Press.