CLASS ASSURANCE KNOWLEDGE ECOLOGY

Jonathan Rowanhill and John Knight
Dependable Computing LLC

Dependable Computing Technical Report TR-2015-1

5/31/2015

Dependable Computing LLC
2120 North Pantops Drive, Charlottesville, VA 22911-8648
www.dependablecomputing.com

Assurance Knowledge Ecology

Abstract

We propose Assurance Knowledge Ecology. By inverting ownership of arguments away from generalist experts and towards domain knowledge experts, ownership and competitive fitness of arguments break down the ad-hoc, expensive, and static nature of system assurance. This model recognizes that domains of engineering already have assurance arguments. They are merely implicit. By encouraging domains to make their knowledge explicit, it can be continuously improved and rendered available to system arguments. Arguments exist and are owned in subject matter experts' domains first, rather than from the needs of individual systems. System arguments, in turn, are derived therefrom and are test cases against domain knowledge. A model of this ecology is presented, and an experiment conducted in explicating domain arguments.

1 Introduction

We argue for the replacement of generic static assurance knowledge with domain-driven knowledge ecology. The purpose of this approach is three-fold.

- *Improved assurance arguments*: which at the present tend to be isolated and underutilized.
- *Improved assurance knowledge bases:* such as argument templates, processes, and guidance, which presently tend to result in ad hoc arguments with low semantic investment in domain models.
- *Improved assurance over system lifecycles*: as assurance artifacts tend to be naturally static without considerable forced activity.

This work critiques the existing organization of assurance knowledge, and presents a hypothesis as to why its current organization has the above weaknesses (Section 2). It argues that a knowledge ecology model reduces these deficits (Section 3). It sets up questions for future research in domain-specific assurance argumentation (Section 4). The work concludes with a discussion of whether such ecology can succeed.

2 A Critique of Current Explicit Argument Techniques

It has long been argued that explicit argumentation is a superior means of assuring system properties. In particular, rigorous system arguments are based on first principles and do not hide information from the reader through transitive trust assumptions (such as occurs with current

standards technology.) In essence, explicit arguments represent advertised knowledge. Knowledge is information that enables action, and explicit arguments improve or enable:

- *Organization* of information in the assurance aspect
- *Improvement* of assurance through induction from real-world systems and environments
- *Reuse* of assurance through induction to create re-usable modules
- *Communication* of assurance through sharing of arguments
- *Improvement of Assurance Theory* in the form of better deduction against rigorous explicit argumentation

On the other hand, the shortfalls of assurance argument regimes are well known.

- *Resource intensive*: Arguments take a long time to build and require many inputs.
- *Large and Complex*: Arguments can become very large and complex for reasonably sized systems.
- *Stale*: Arguments do not evolve in response to new information about systems
- *Ad hoc*: Arguments tend to be built from the ground up as semantically rich arguments, or built from generic templates with compromised semantic depth.
- *Sub-optimal*: System arguments can be un-reusable, inscrutable to readers, unconvincing, and fail in other measures of importance to stakeholders.
- *Begrudging*: Arguments are only created because they are required.

Many attempts have been made to create templates and modules for arguments to overcome issues of scale and cost. Management systems attempt to enforce arguments as living documents tied to a system's lifecycle. Still significant effort, energy, and enforcement of policy are required to overcome or compensate for the above limitations. Why? To address this issue, consider the relationship between knowledge and argument below.

2.1 Typical Knowledge Interaction in Building Arguments

Error! Reference source not found. demonstrates a typical relationship structure surrounding an assurance argument for a system. To the left is an engineering domain contributing to the argument (real systems have many). To the right is the domain of expert knowledge in building explicit arguments. In the middle is an *ad hoc* domain that comes into existence in order to support the argument.

The left-hand engineering domain owns knowledge about engineering in the form of design, implementation, validation, and verification procedures, templates, and guidance. Never the less, the actual argument as to why this is sufficient is generally implicit.

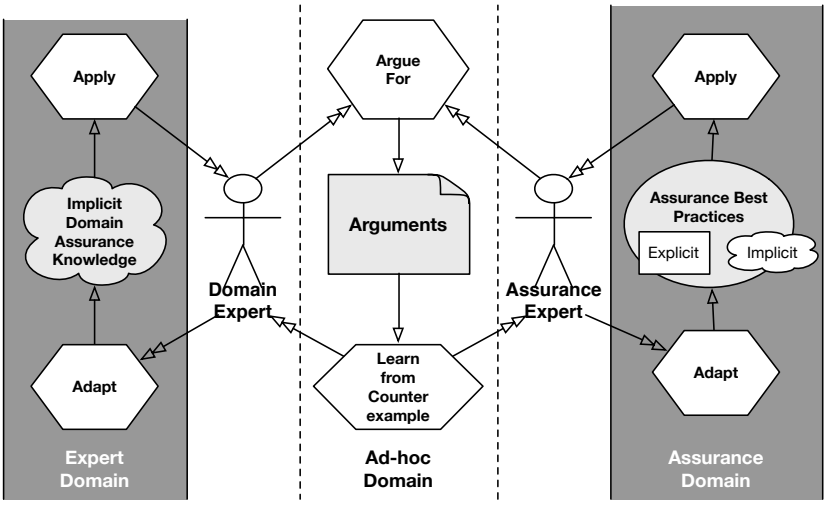

Figure 1. The Ad hoc domain of typical assurance arguments.

The right-hand domain owns knowledge about how to write sufficient arguments. This includes argument procedures, templates and guidance material for best practices. There is also implicit knowledge that assurance experts have failed to capture explicitly.

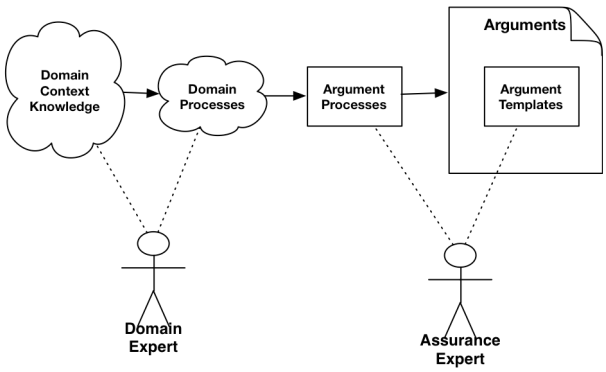

Figure 1. Conforming domain knowledge to argument standards.

Figure 1 shows a typical example of this model in action. Domain experts apply context and processes from their domain. They communicate this information to assurance experts. The domain knowledge is then transformed through argument processes of the assurance expert. These processes *capture* the domain knowledge, *conforming* it to the semantic and syntactic constructs of the argument expert's domain. The result is an argument driven by the concepts of best practice from the generalist perspective.

2.2 Issues of the *Ad Hoc* Domain

At first glance, the above model of collaboration appears successful. Experts bring their respective expertise to a shared space and create new knowledge. But why was this knowledge created, how is it maintained, and how does it encourage further new knowledge in its respective domains? As we will see, it was created by mandate, it is maintained by regulation, and it doesn't encourage further discovery and learning. As a result, arguments appear begrudgingly, and the results are stale and sub-optimal. Their complexity is never managed, and they require massive commitment of resources each time one is created. Enforcement is then applied, using yet more resources to try (and often fail) to overcome these limitations.

Significant limitations occur because of the creation and continued existence of the ad hoc knowledge domain. These problems are roughly categorized as problems of ad hoc semantics and ad hoc ownership:

2.2.1 *Ad Hoc* Semantics

The *ad hoc* knowledge domain combines assurance knowledge from multiple domains in an *ad hoc* domain. The resulting domain has the following problems:

- *Translation Barriers*: Knowledge from expert domains must be transported into the generalized *ad hoc* domain. To do so, semantic barriers must be crossed. This is illustrated in Figure 2. Domain experts interpret the templates, guidelines, and other resources of general assurance. They then attempt to elicit their implicit knowledge in a form compatible with these generic resources. Some semantic translation then occurs to

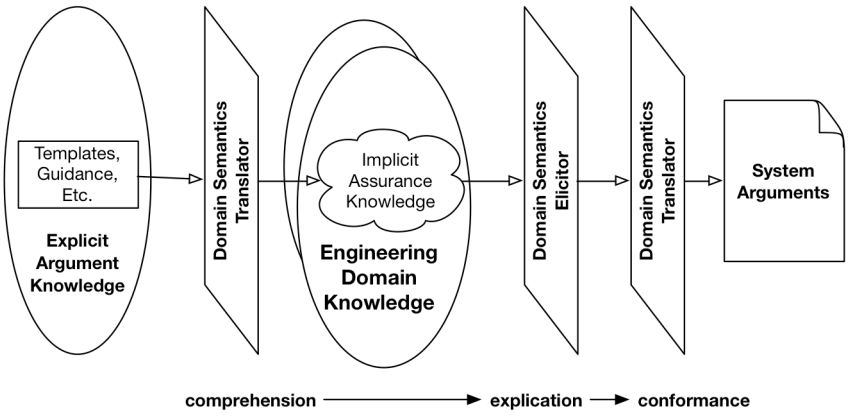

Figure 2. The semantic translation process in ad hoc arguments

the expectations of the system argument space. The result is what is written in a system assurance argument. The same semantic barriers exist for information flowing in the opposite direction. Counterexamples to an argument emerge if its argument is flawed. This information can be applied to improve the system argument, but must cross semantic barriers to affect domain knowledge.

- *Recurring Semantic Fusion Work*: A system is assured through presentation of expertise from many domains in the shared *ad hoc* domain. The problem of combining domain knowledge re-emerges each time a system is built. Which domains dominate argument structure and evidence? Which can be combined? These are difficult questions with expensive answers each time they occur.
- *Ad hoc Bias*: Which experts from domains and assurance have contributed their knowledge to the resulting argument artifacts? Is it the best knowledge as far as the domain community is concerned? Is it well represented? How up-to-date is the knowledge that was applied to build the argument? An ad hoc domain contains an ad hoc result.
- *Confirmation Bias*: When a viewer from one domain reads argument from another domain, it is viewed as correct by default due to the complexity and apparent work put into it. In an ad hoc domain, non-domain participants tend to suffer from confirmation bias about the parts from other domains. A better verification and validation process would draw more review from respective expert domains and less from the *ad hoc* domain.
- *Implicit Conformance*: Arguments, as artifacts, tend to become monolithic entities representing an implied consensus, whether such a consensus exists or is merely an artificial conformance to the ad hoc domain.

2.2.2 *Ad hoc* Ownership

Ownership of *ad hoc* domains is ad hoc. This has two important properties:

- *Unclear Ownership*: Who owns the *ad hoc* assurance domain. Do the experts own the domain? Do the system owners own the domain? Who manages the argument content over a lifecycle? Where do authority and the role of delegators lie? If these questions result in informal answers, then mistakes often are made in the creation and maintenance of arguments. Where the relationships are formalized, authority and subservience are artificial. Where peer groups are formed, conflict can result in sub-optimal knowledge development.
- *Once-Removed Stakeholders*: The experts of various domains truly care about the knowledge base of their domains. The *ad hoc* knowledge base of the argument domain is one step removed therefrom. Thus, human nature tends to lead to experts considering the *ad hoc* explicit argument domain to be of secondary importance to the implicit arguments of their respective disciplines.

2.3 Resulting Problems affecting Arguments

As a result of the above issues with the ad hoc domain, the contained argument has the following negative attributes:

- *Isolated*: *Ad hoc* semantics isolate an argument from important communities of practice, comparison with other systems, and outside experts.

- *Static*: *Ad hoc* semantics and ownership make difficult the induction and deduction of improvements from counterexamples and improving domain theory, respectively. The argument tends to become a static artifact by nature, with enforcement and energy required to keep it as a living document.
- *Diluted: Ad hoc* ownership leads to reduced precision and adherence to the theory and practice of each domain. The net argument loses resolution with respect to each domain's implicit knowledge. Theory of general argumentation may go unused or over applied. Templates do not adapt to the needs of knowledge domains, as they remain defined by general theory within the assurance domain.
- *Error Accepting:* Semantic barriers and fusion issues can lead to errors of translation and exclusion. Ownership issues make the authority and (more importantly) responsibility to correct errors unclear. Once removed stakeholders make the desire to correct errors less self-motivated as compared to correction of domain knowledge.
- *Suboptimal*: Once isolated, it is hard for knowledge to compete to improve the standing of best practices.

Unfortunately, *isolated*, *static*, *diluted, error accepting, and suboptimal* arguments are the norm. When an error is detected in a body of knowledge, expert domains are updated but this rarely translates to existing system arguments. When counterexamples to existing systems are detected, they are often not translated into meaningful assurance argument changes. When they are, they often stop there, not making there way back to domain knowledge as a contribution.

Given the above critique, it is concluded that the organization of knowledge and arguments is impeding their potential. A new model is required that changes knowledge relationships. It should not enforce artificial relationships in *ad hoc* domains with *ad hoc* ownership. It should emphasize expert knowledge within expert domains, where engineers are self-motivated to apply improve their trade.

3 A Model of Assurance Knowledge Ecology

We hypothesize that if one modifies the expected relationships of assurance knowledge then some of the negative characteristics of system arguments will improve, without the need for significant enforcement regimes. The form of these new relationships is *assurance knowledge ecology*.

3.1 Towards an Assurance Knowledge Ecology

Much can be learned about how to build knowledge ecologies from existing models. In this work, we consider two models. The first is a fundamental model defining the concept of knowledge ecology. The second is a model derived from the particularly successful knowledge ecology—open source software.

3.1.1 The Malhotra Knowledge Ecology Model

Knowledge ecology is a model of information systems that deemphasizes the enforcement of rules, fixed organizations, information models and processes, and replaces them with:

- *Self-Regulation*: Knowledge ecologies focus on actions that stakeholders take to build, protect, and improve knowledge in which they have vested interests.
- *Self-Organization*: The organization of knowledge ecologies is adaptive and driven by the needs of communities that take ownership of information,
- *Competitive*: The knowledge in knowledge ecologies is consistently discovered, selected, measured, compared, tested, and adapted by stakeholders. As knowledge is information that can be acted upon, this includes both information and best practices.

This contrasts with traditional approaches to knowledge management. Consider again the example of **Error! Reference source not found.**. Argument experts contribute a template library of 'argument patterns' to the argument artifact. They also provide processes, sometimes informally, other times more rigorously, to help 'fill out' the argument. The intrinsic assurance knowledge of engineering domains is made to conform to assurance templates and processes. It is transcribed in an *ad hoc* manner into an argument, where it sits untested except against the argument as a whole and is updated by mandate.

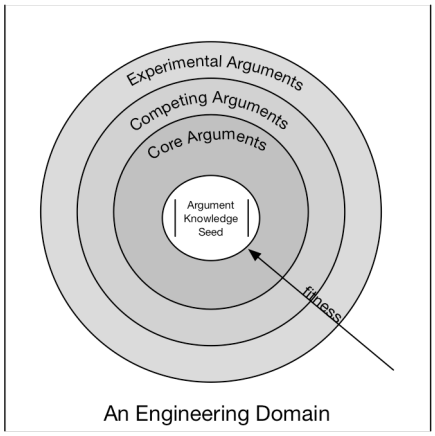

Figure 3. Arguments as competitive knowledge within domains of expertise

In contrast, Malhotra's knowledge ecology model would discourage conforming to generic argument templates and assurance processes. Instead, as shown in Figure 3, domain experts would be empowered with seed knowledge about explicit argumentation. From this seed, they would construct models of explicit assurance within their fields. Competing models within a community would be assessed for their utility, robustness, readability, believability, and other qualitative metrics. Thus, the theory and trade of domains would now include a theory and practice of explicit assurance, continuously improved by selective pressure. In this model

- *Model building* for explicit assurance occurs and is owned within domains.
- *Natural motivations* that regulate domain knowledge now regulate assurance arguments based on their *fitness* determined by the *community of practice*.

Assurance arguments are fit and adapted within their domains of origin. Once they are applied to system arguments, they are under constant test against real world conditions for the systems they assure. This is depicted in Figure 4. In this organization

- *Systems are sensors* that continuously test domain knowledge.
- *Domain argument ownership encourages improvement,* as domains will react to argument failures as failures of their domain knowledge.

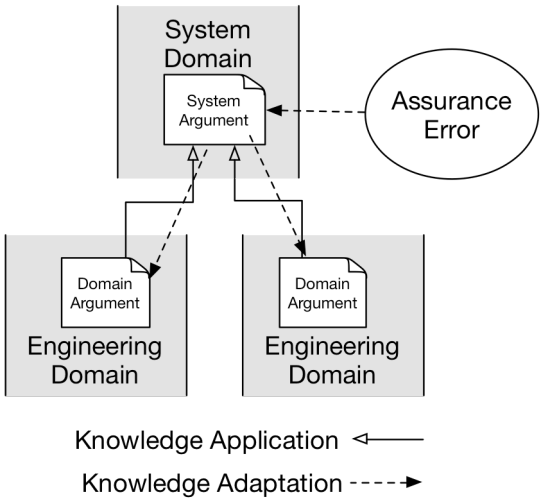

Figure 4. Application and adaptation of

Systems test the applicability of domains arguments, their ability to blend with arguments from other domains, and the accuracy and completeness of assurance in the field. For example, failures of assured system properties are counterexamples to the legitimacy of an argument. System argument failures transfer onus to one or more domains, which for the sake of their own integrity, must solve failure cases. Thus, this model encourages continuous testing and improvement of both system and domain assurance knowledge.

This model moves away from artificially enforced management of system assurance towards a constructivist approach to argument knowledge. Domain expertise and pride of practice is a self-motivator for the cultivation of competitive explicit assurance argument within domains of engineering. What is needed is the proper technology and artifact representations to achieve self-sustaining assurance knowledge ecology.

3.1.2 Lanzara-Morner Open Source Software Ecology Model

Lanzara and Morner identify open source software projects as existing in knowledge ecology. Its key components are artifacts, computer mediated interaction, processual interplay, and counterbalanced innovation and stability.

Artifacts:

In open source ecologies, source code is a primary inscription of human agency and knowledge. It can be measured for qualitative and quantitative fitness. Wikis and discussion threads capture, archive, and promote discussion and recording of key points and decision-making surrounding the software.

Three key information artifacts for assurance knowledge ecology would be:

- *Explicit Arguments*: An argument artifact can take on a written form or a graphical form such as Goal Structured Notation (GSN). An argument includes detailed evidence to justify the claims of the argument.
- *Processes*: Best practices for interacting with arguments are represented as explicitly written processes. These processes can be text, as in many open source communities, or represented in a graphical workflow notation.
- *Guidance*: Guidance material general appears as tutorials and explanations in written text form. These text documents can be in many digital document formats as presently available.

The core resources are surrounded with context resources.

- *Discussion threads* help orchestrate the management of resources.
- *Written articles* interface the domain with external domains and communities.

Computer mediated interaction:

Technology must be present to mediate information flow. Technology dictates process in traditional knowledge management infrastructure. This is relaxed in knowledge ecology. Specific tools closely support information, but the tools are not tightly bound to one another. Instead, the tools are used in whatever combinations meet current best practices.

Tools that will support assurance knowledge ecology include:

- Argument authoring and view tools
- Evidence gathering tools
- Text authoring tools
- Discussion tools to support conversation threads
- Issue tracking tools

Conversation tools for threaded communications would be critical for surrounding arguments with constructive discussion. Finally, wikis in which the theory and purpose of arguments could be recorded are an important gateway for users to find and interface argument knowledge.

Knowledge ecology does not enforce tool use, but works efficiently as users discover and apply preferred tools. For example, version control and continuous integration systems enable rapid yet loosely coupled interaction between participants. Thus they are consistently present in open software ecologies.

Processual interplay

Knowledge ecologies have many overlapping activities that emerge and fade over time as ideas are processed and developed. In assurance knowledge ecology, we see this as primarily *communities of practice* emerging within disciplines to master and develop assurance arguments and their support infrastructure. Systems will serve as a means for communities of practice to interact with one another.

In addition, we anticipate domains working together to solve problems of semantic fusion in a consistent and reusable way. This would likely occur where a class of systems has commonly overlapping assurance domains, or where disciplines naturally overlap in their work.

Counterbalanced innovation and stability

Knowledge ecologies attempt to balance innovative forces with stabilizing forces. Innovation occurs through the competition of knowledge artifacts. Stabilization typically arises through the organization of a community and its authorities.

Lanzara and Morner found that key stakeholders exhibit regulatory positions within a community of practice. The organization of the typical community of practice is shown in Figure 5. Core contributors are a small but powerful group. As delegates of authority, they act as filters on changes, and they act as directors for future efforts. This group consists of key stakeholders. More peripheral members, who make passing contributions and are still learning the domain, interact with the core group to check one another's models of the domain (sometimes contentiously) and contributions. Our own observations suggest users are still more numerous and do not modify the system. Finally, observers tend to just watch the space.

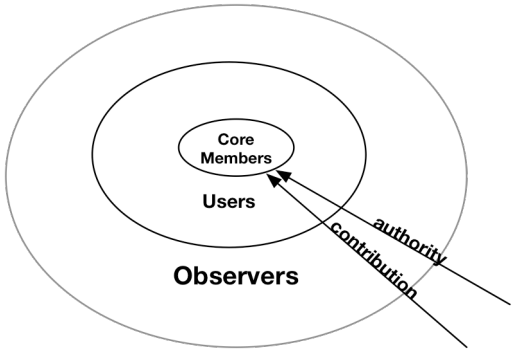

Figure 5. Typical population and authority distribution in an open source software community

In assurance knowledge ecology, special attention will need to be paid to regulatory regimes. Regulation is driven by enforcement and incentive, and is therefore a traditional knowledge management in contrast with knowledge ecology. In traditional models, regulators accept or reject assurance knowledge domains. In a more ecological model, regulators are active participants in communities of practice for specially recognized domains. The former role serves traditional regulatory purposes. The latter enables regulators to help shape and understand an adaptive assurance regime.

3.2 An Assurance Ecology Model

From the principles and synthesis above, an assurance ecology model is presented. This model contains identification of the necessary assurance information, and models it in actionable forms. We have identified, procured, and implemented technology to enable the ecology to operate. Finally, the model includes a hypothesized emergent social organization.

3.2.1 Information Organization

Key Information Objects

The key information objects of the ecology, shown in Figure 6, are:

- *Resources*: Resources are bundles of artifacts the codify domain assurance knowledge. They include argument templates, guidance materials, useful processes, and tools.
- *Assurance Cases*: Assurance cases are arguments for the assurance of system properties. They consist of combined arguments, explanatory text, and accompanying evidence documents. They represent codified system assurance knowledge.

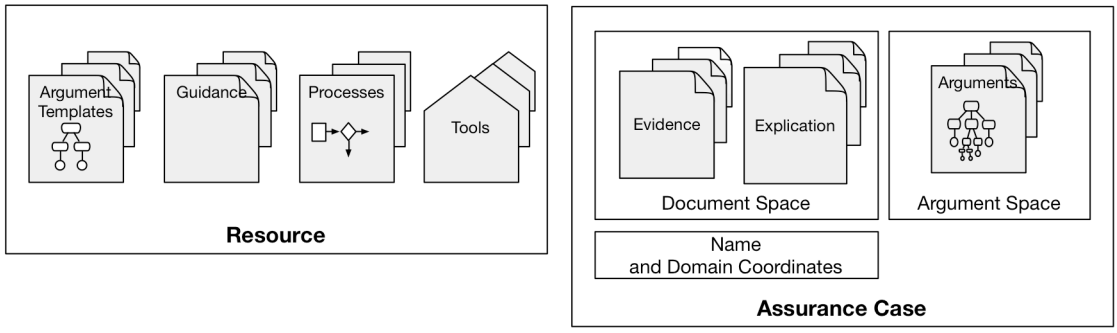

Figure 6. Primary Information of Assurance Knowledge Ecology.

In more detail, the artifacts found in a resource are:

- *Argument and Document Templates*: Representation of domain assurance arguments are made explicit in templates. For the purposes of our model, Goal Structured Notation (GSN) is promoted as a shared standard. Other standards might emerge and dominate. Evidence in support of the argument's claims is referenced from the argument proper and

is generally presented in document form. Evidence forms and other useful information are presented as document templates.

- *Guidance*: Guidance consists of general helpful information for use of the resource.
- *Processes:* Processes for creating arguments, making decisions, and collecting evidence are useful to help define best practices. Many knowledge ecologies represent these informally within group rules, perhaps written on a wiki. We contend that representation of best practices with rigorous process definitions is a useful for assurance. As such, we encourage the inclusion of processes in Business Process and Modeling Notation 2 (BPMN2). These are in a well-understood format. It is also possible to execute these processes on a workflow engine to help teams orchestrate complex actions as well as present auditable evidence of process to regulators
- *Tools*: Specialized tools for a domain activity are expected. These could be complex simulation codes or simple checklists. They are useful leverage in applying a resource.

Projects

Secondary to resources and assurance cases are means objects. They are the information that supports the lifecycle and more fine-grained management of resources and assurance cases. As with open source software, we have selected a project model for this information. Projects are depicted at the top of Figure 7.

A project consists of

- *Goal*: The goal of the project
- *Name and Domain Coordinates*: The name of the project, and a string defining the domain to which the project belongs.

Resource projects are either a blank slate or can reference an existing resource to adapt or tailor. System argument projects also include:

- *Chosen Resources*: Already existing resources selected to help accomplish the project
- *Issues*: Issues that must be resolved to complete the project

Tertiary information is information surrounding the management and lifecycle of projects. This includes context documentation for projects, issues that exist or have existed in the project, discussions about the project, and alerts that have been raised on the project or received about related resources.

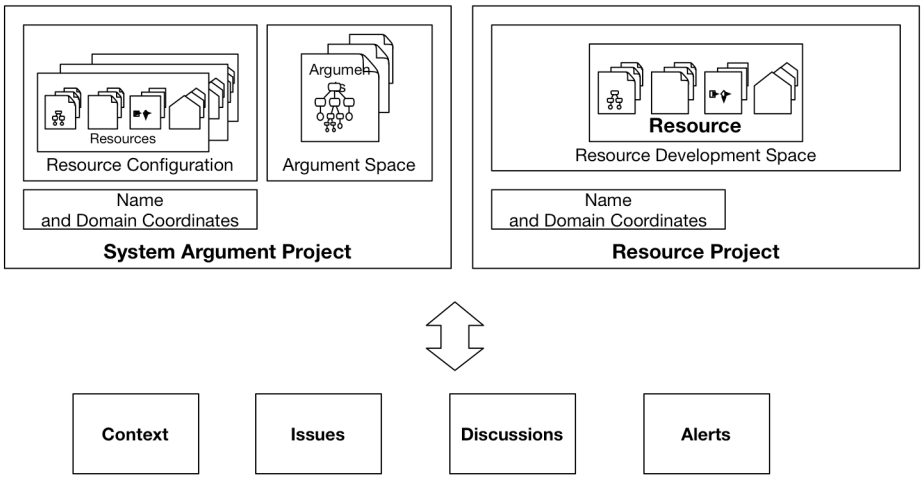

Figure 7. Secondary and Tertiary Information of an Assurance Knowledge Ecology

Metrics

The metrics of resources and projects are an important part of the competitive model of knowledge ecology. For assurance arguments, some of the core metrics are:

- *Strength*: The extent of the argument to adequately argue its desired property assurance.
- *Confidence*: The confidence of the community in the strength of the argument.
- *Readability*: The ability of the argument to be easily read by interested parties.
- *Accessibility*: The ability of stakeholders to make a meaningful assessment of the arguments strengths and weaknesses.
- *Adaptability*: The ability of an argument to be successfully adapted to the range of conditions seen in system environments.
- *Coherence*: The ability to be applied in semantic merging with other arguments, possibly from other related domains.
- *Modularity*: The ability to be applied in parallel with other argument templates.
- *Representativeness*: The extent to which the argument represents the best thinking its domain(s)
- *Successfulness*: The extent to which the argument has proved successful over time in use by systems. Components are known accuracy and completeness.

It is expected that the relative merits of arguments will be judged and compared on these and other factors. Similar metrics can exist for process descriptions and guidance, but are beyond the scope of this present work.

3.2.2 Technical Components

The knowledge ecologies technical component consists of tools supporting the projects and repositories of the ecology. Thus, a technical hub for a knowledge domain can be created by instantiating a resource repository and a project server at the hub. This model deliberately replicates the successful open source community model. This is illustrated in Figure 8.

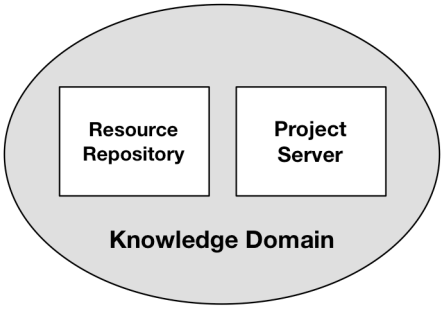

Figure 8. The Technical Services of an Assurance Knowledge Domain.

The resource repository organizes latest in a domain's assurance knowledge. The project server manages assurance knowledge over its entire lifecycle.

Resource Repositories

The assurance knowledge ecology implements a resource repository as the collection of several services, as presented in Figure . The primary service is that of an artifact repository. Our implementation of the technology utilizes a maven-compatible artifact repository. Examples include the Nexus repository system or alternative freeware solutions. These repositories can be uploaded to, downloaded from, browsed, and searched. They organize resources through resource coordinate names that can be used to group resources by domains, specializations, ownership, and affiliations. Recall that all resources have such a coordinate name as specified previously.

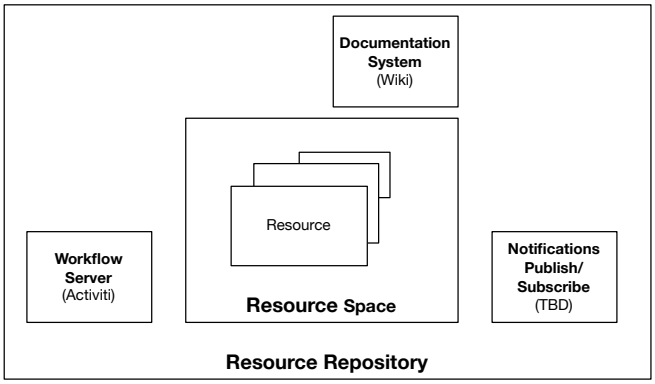

Figure 10. An Assurance Knowledge Resource Repository

In addition to the Maven repository, our model includes a workflow engine. It is here that BPMN2 models can be executed to enforce community standards for submitting and updating resources. It is also a starting point for launching community processes to deal with system counterexamples and detected failures concerning the resources of the repository.

A publish/subscribe node exists alongside each repository. This network is used to *publish notifications* about changes or flaws related to resources. Subscribers can be members of communities of practice, or downstream users of domain content, such as system administrators and developers. They will receive messages about argument failings.

Finally, documentation systems exist in concert with the repository, where information about the domain itself can be represented. Here, users can find out what the domain represents, and learn more about the makeup of the community in general.

In our current implementation of this ecology model, Dependable Computing maintains a resource repository for seeding general assurance knowledge. It is called the *General Assurance Repository*. It includes a resource to set up one's own resource repository for a domain, as well as resources containing knowledge about general explicit assurance argumentation. It contains resources to start and maintain projects as well. This includes a distribution of the Project Server, as discussed below.

Project Support

Projects are created and exist within a Project Server, illustrated in Figure . A system server is a collection of co-located service tools, including a version control system (Git is currently used in our implementation), an issue tracking system, a workflow system to orchestrate and enforce BPMN2 processes, a directory of projects, and surrounding communication technologies such as wikis and discussion platforms.

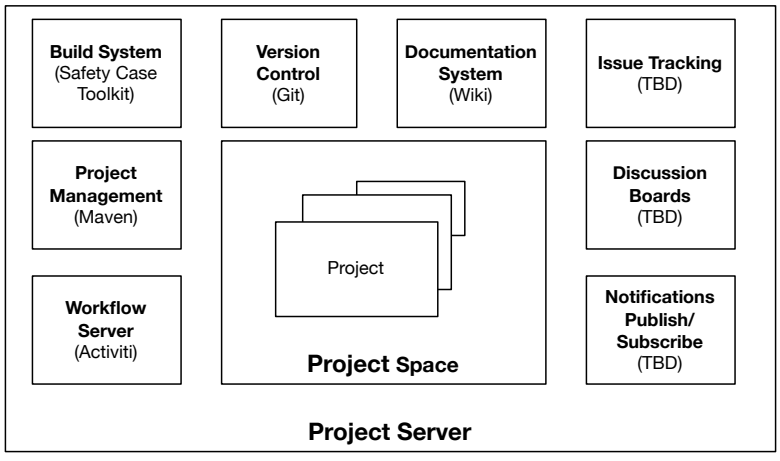

Figure 11. Project Server for Management of Assurance Knowledge Project

The project server supports the creation or installation of many simultaneous projects, so that it can be thought of as a community hub for a domain's project activities. The server supports both system and resource projects.

Projects are defined using maven specifications (pom files). Two project templates are supported, one for a resource project, and another for a system project. Users define their own project types as needed, either creating new models or refining the existing models provided by the central assurance repository.

Resource projects allow one to define or refine a collection of resources. Thus, they support building and testing argument templates, workflow processes, guidance documentation, and tools. The project server builds projects, tracks issues, and supports discussions.

System projects consist of two key pieces:

- *A collection of resources*: Resource modules chosen from various engineering domains for their relevance to the project and their fitness according to members of those domains.
- *Template assurance argument space*: This is a directory structure in which explicit assurance arguments for the system and accompanying evidence reside.

By setting up a Maven 'pom' file for the system project that points to the required resources, Maven will *fetch* and install the assurance resources from their respective resource repositories. Recall, any domain can have a resource repository, and a project may pull resources from many domains of assurance knowledge. Also recall that the resources consist of argument templates, processes, guidance and tools. The Project Server organizes and makes available the argument templates, guidance, and tools. It also installs and enforces the processes that come with the resource bundles.

In summary, a system project is a configuration of assurance resources. This configures documents, processes and tools that will be used during the lifetime of the system--from creation through maintenance and retirement.

Authoring Tools

General document authoring tools of the users choice can be used to build guidance, and appropriate viewers are required for the various common document formats prevalent in an information system (word, mmd, pdf, html, *etc.*)

Included in our general assurance package are GSN authoring and reading tools. These promote the use of a Goal Structured Notation for the development of argument templates in resource packages as well as for system assurance arguments. Wikis, discussion boards, and issue tracking systems have their own authoring systems that vary with implementation, but are generally sufficient for use in knowledge ecology.

A Note About Workflow Technology

As mentioned above, both projects and resource repositories utilize workflow technology. Many knowledge ecologies operate on human orchestration alone. They do so to avoid burdening

activities with *a priori* assumptions about the nature of practices for a community. This also prevents best practices from becoming static and obsolete.

However, property assurance requires doing the right things at the right time, so that the resulting arguments are strong. For this reason, we believe a rigorous process language, such as BPMN2, is as valuable to the ecology as a rigorous argument language such as GSN. Such language is clear and standardized. It is data, and therefore can be rewritten, adapted, thrown out and replaced, *etc.* It is an artifact that can be clearly understood by a community, as well as executed by a workflow engine. Thus, just as arguments are measured and selected, so to can best-practices as represented in executable process diagrams.

Together, the information-artifact and executable qualities of workflow language elements should make a basis for knowledge ecology process that is both adaptive and sufficiently rigorous as to be compatible with regulatory requirements. Simple processes diagrams might be of conversational value in some communities of practice. In other, more critical assurance areas, they might be the subjects of strict adherence and process-engine enforcement.

3.2.3 Social Organization

Social organization resulting from the above information and technical organization is expected to resemble that of open source software. Where we expect it to differ is in a more important role for knowledge canon in domains, and in the role of regulators within domain communities. An

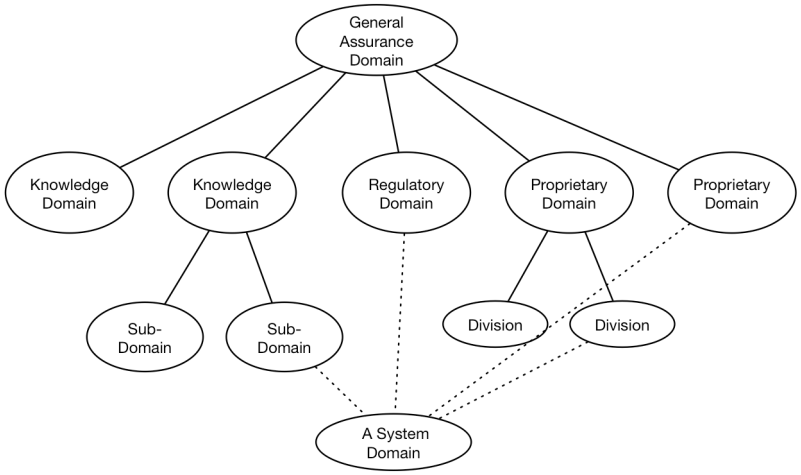

Figure 9. Expected hierarchical nature of assurance domains.

example of the overall social structure is depicted in Figure 9.

Knowledge, Proprietary, and Regulatory Domains

A discussed previously, the primary social organization of the assurance knowledge ecology is the knowledge domain. This is an area of expertise that influences the development of systems. It can be an area of engineering, science, or art that impacts a property of a system for which the assurance thereof can be argued. The extent of rigor available to the argument will depend on the

domain. It is expected that the knowledge domains that will benefit most from assurance knowledge ecology naturally will be those for which the biggest stakes in assurance appear, such as for safety and security. However it is counter to knowledge ecology to circumscribe range of use.

Second to knowledge domains, but also very important, are Proprietary Domains. Proprietary Domains, such as corporate entities, throttle the flow of information in and out of their domains as required by corporate interests. Proprietary groups are common in open source software ecologies, and would likely be important in assurance knowledge ecology as well.

A third potential domain is a regulatory domain. These domains focus on regulator policy and serve as a community of practice for the regulatory stakeholder. Here, arguments that become standards can be developed and adapted. Relationships with discipline and proprietary domains can be developed relative to these 'home bases'.

Communities of Practice

Every live domain requires at least one community of practice. We envision the communities of practice being similar to those of Figure 5, with a significant overlaps in role and a larger role in

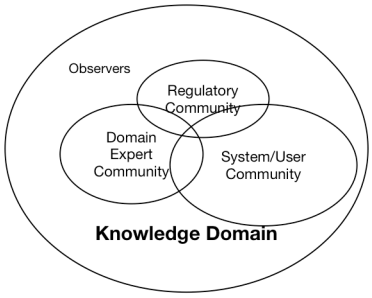

Figure 10. Community of Practice of an Assurance Domain

general for regulators. This is illustrated in Figure 10. The stakeholders of the domain are likely to belong to the following overlapping sub-communities:

- *Subject matter experts*. These are the fundamental owners of a domain's models of assurance. Their stake is the reputation and use of the field.
- *System builders*. These are the *users* of software communities of practice. Their stake is the success of their systems.
- *Regulators*. Their stake is the quality of the assurance of system properties.
- *Observers*. These are outsiders from other communities. Their interest is peripheral and likely to be driven by knowledge discovery and learning.

Domain Hierarchy

Domains may contain sub-domains, each with its own community of practice. Likewise, domains might overlap. Where this occurs there might be natural competition or coordination. Or

there might be mutual ignorance! The latter is a problem not directly solved by an emergent ecology.

The resulting domain organization is somewhat hierarchical with important sibling relationships. At the root of a domain hierarchy is general assurance knowledge. We currently envision this as being a seed repository similar to the one maintained by Dependable Computing. Beneath this are knowledge domains, proprietary domains, and regulatory domains. These sibling domains are peers with different roles to play in one another. Their respective communities of practice as discussed above, tend to exchange representatives. Underneath domains are sub-divisions that should occur naturally.

System domains are still *ad hoc*. However, they take their argument resources directly from explicitly stated resources within domains of knowledge. The community of practice for a system is its stakeholders, whom rely upon domains of expert knowledge in various combinations. Furthermore, classes of systems can themselves become knowledge domains, so that knowledge can accumulate as tradecraft for similar works. The key point to take away is that system are now sensors for the improvement of domain knowledge.

Domain Dynamics

It is important to remember that nothing about domain organization is static. Domains exist so long as they have value to stakeholders. Some domains might be experiments while others represent long-lived disciplines. Still other domains might be industry self-regulation attempts and others more institutional regulation. The marketplace of domains is dynamic.

Information Flow

The information flow of knowledge within the assurance knowledge ecology should differ from that of present argument development. In particular, it should reduce the semantic barriers present in today's argument construction, and should maintain live argumentation both within domains and systems. What we expect to occur is presented in Figure 11. First, we note that the primary means of inter-domain communication is through knowledge sharing discussions. There might be competition and knowledge contention in these discussions. These are mediated by the explicit arguments that can be shown to one another's domains.

The interaction between expert knowledge domains and system assurance domains is of particular interest. Systems pull assurance knowledge resources from expert domains as defined in their project configuration. If something should be wrong with a resource, then the detector—whether system domain or expert domain—alerts the related domain accordingly.

Independently, systems are under pressure to resolve any failures of assurance that occur. These might occur because a resource cannot be applied to a system, or because the system fails to maintain a property that was assured by its argument. Such failures raise alerts within the system and within related expert domains.

Expert domains are under continuous adaptive pressure. This might be due to competing models within the domain, or in response to reported system failures. In either case, changes to domain

knowledge made concrete are published as alerts to systems, so that their argumentation can be adapted accordingly.

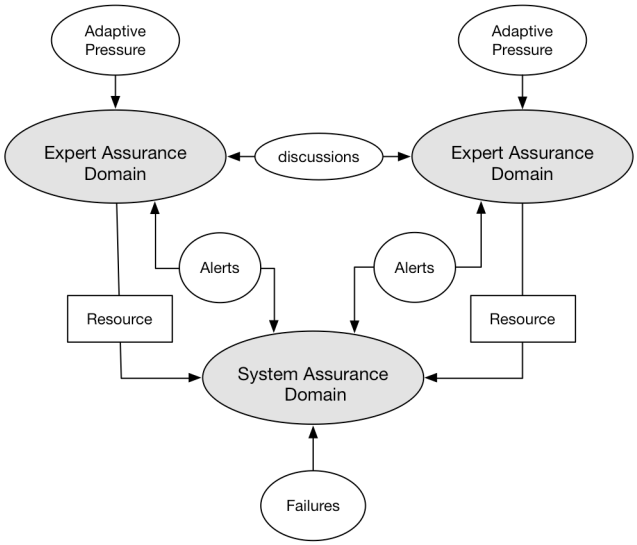

Figure 11. Information flow across domains.

4 Future Work: An Experiment in Domain Specific Assurance

Is explicit argument using Goal Structured Notation a viable product within a specialized knowledge domain? How does such an artifact interact with the experts in the domain? Is it redundant? Does it point out areas of confusion or conflict within the community? Is it something that can be applied to systems, and if so, with how much semantic translation to the system context?

In order to begin to answer these questions, an experiment was performed in which domain-specific argument models were built for the area of avionics flight control software. This is an area above strict avionics control theory, but more specialized than general software development. It is an area that relies heavily on simulation in order to argue for its desired properties. What would the benefit of making this implicit argumentation explicit be for the development of a flight control software system assurance argument?

In the next iteration of CLASS, we will conduct experiments to determine if domain specific argumentation can be captured, and how successfully such models can be applied to system argumentation. If successful, domain-owned resources will become the key driver behind the CLASS Resource Repository model.

5 Conclusions

We believe that the reason arguments have become overly costly, static, and un-modular is because they failed to explicitly recognize the role of domain knowledge in assurance. They assume it is more easily semantically translated to general philosophical argument structures than it truly is. The resulting arguments area also politically divorced from knowledge ownership of the domains on which they critically depend. By reversing the direction of information flow in the development of arguments, we believe knowledge ecology of domain-owned explicit argumentation will improve assurance argument regimes.

The assurance knowledge ecology, as presented, is fully implemented at the technical and information artifact levels. Our experiments demonstrate the viability of explicit domain argument knowledge. However, what remains to be demonstrated is the value of such arguments to domain stakeholders, which is the critical property that would be needed for the bootstrap of such ecology to result in its self-propagation.

Another important area of research lies in argument metrics. These are properties of arguments that qualify represent their value to engineering domains. A set of arbitrary metrics was provided in this work as an example. Significant research work could be done on the discovery and analysis of metrics and their value.

It remains to be seen if knowledge ecology can be ‘seeded’, as proposed in this work. Many knowledge ecologies emerge into existence from known, more limited practices and resources. We argue that the similarities between argument and open source software make the potential real. However, it can only exist if argumentation is deeply valued by communities of practice. Such adoption remains to be seen.

To this end, future work will seek to aid in the development of knowledge and regulatory domains utilizing this framework. The opportunity will be for communities of practice to come together and express their concerns and beliefs about the future of dependable systems. After all, expert input *is the knowledge* of the ecology, and is necessary and critical for it to thrive. Of great important are contradictory and competing views, as well as new and diverse experience.