
# CLASS SERVER TOOLSET: DESIGN AND IMPLEMENTATION

Jonathan Rowanhill
Dependable Computing LLC

Dependable Computing Technical Report TR-2016-1

5/31/2016

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

# CLASS Server Toolset: Design and Implementation

**Abstract**. The goal of the Comprehensive Lifecycle for Assurance of System Safety (CLASS) is to obtain assurance for relevant elements of a domain over their lifecycle. The Server Toolset provides an infrastructure that supports all of the information in a domain, i.e., an area of knowledge. The notion of "support" in this case means supporting: (a) all aspects of information management, (b) all phases of the system lifecycle, and (c) all elements of assurance, including the CLASS concept of synchrony. The toolset combines state-of-the-art software project management and knowledge capture and management technologies with custom software and powerful tools developed elsewhere.

## 1 Introduction

The CLASS Server Toolset have been designed and prototyped in support of the Comprehensive Lifecycle for Assurance of System Safety. This report describes the tool design and prototype, relating them to the design principals of the CLASS model.

The conceptual abstraction of CLASS models the *assurance* of a *domain* over its *lifecycle*:

- *Domain*: A domain is an area of knowledge. A domain might contain reusable knowledge such as the basics of software engineering. Alternatively, a domain might be highly specialized, such as one being concerned with the specification, design and implementation of a particular system.
- *Lifecycle*: The stages of use of a domain. A lifecycle consists of phases, each of which is distinguished by the primary activities taking place between the domain and its users.
- *Assurance*: Assurance is the inductive, best effort computation of belief in stated properties of a domain's elements. CLASS applies argument that properties hold for stated elements of a domain.

The goal of CLASS is to obtain assurance for relevant elements of a domain over their lifecycle. This general picture is shown in Fig. 1.

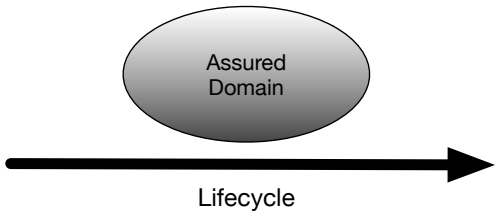


Fig. 1 CLASS conceptual view.

For example, if a domain contains basic software engineering knowledge, then the purpose of CLASS is to assure properties of this knowledge over its lifecycle. In this case, assurance

  

might take the form of confidence arguments that can be applied by other assurance arguments in other domains, as well as being used as a means of judging the fitness of a domain's knowledge. For system domains, CLASS assures properties, such as safety, over the lifecycle of the system. These have direct applicability to real world application of the system.

This high-level picture of what the CLASS tools must do is then refined into a set of mechanisms and actions that can be implemented as a coordinated service between participants in a domain and its assurance.

# 2 High Level Design

The high-level design of CLASS, as presented in the report body, consists of a feedback loop between actions on a domain and monitoring of the domain to support its assurance. This perspective is illustrated in Fig. 2. An *assured domain* is a domain for which CLASS monitors actions performed on the domain such that they either 1) keep the domain assured, or 2) result in reactions that repair domain assurance. Evidence of assurance arises from the static properties of CLASS activities and/or from the monitored state of the domain. This evidence is fed to the assurance of the domain.

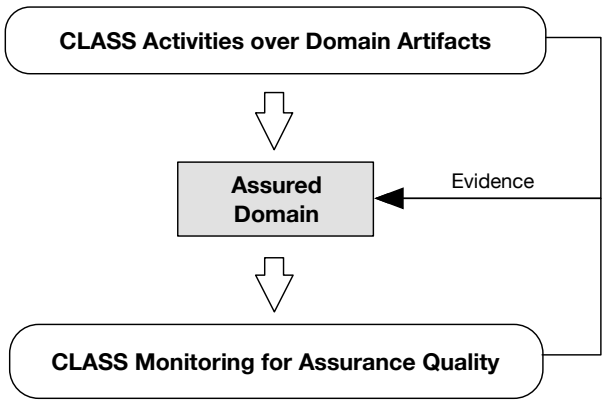


Fig. 2 CLASS high-level design.

The high-level design of CLASS is derived from an understanding of domain artifacts, and the functions performed between them over the domain's lifecycle. In order to further refine this model, we must consider what kinds of elements exist within a domain, and what kinds of actions and monitoring can be performed on it, using proven tools and technology and state-of-the-art assurance methods.

## 2.1 Assured Domains

As described previously, a ***domain*** is a body of knowledge shared in a collective understanding by a community. It is not equivalent to the artifacts produced from it. The domain includes knowledge that has not been recorded as an artifact outside the understanding of knowledge possessors.

An ***assured domain*** is one where best effort is made to assure rigorously stated properties about the domain's knowledge. An assured domain is in a state of ***synchrony*** between the

  

domain's contents and their assurance. Therefore, domains are unlikely to be fully assured over time. Many domains cannot be fully assured at any time. However, the relative quality of assurance for a domain is an indication of its ***assurance state*** as viewed by CLASS.

In this model, a *system* is a domain in which the implemented provision of defined services is of particular concern. It has its own knowledge specific to the system, and imports knowledge from other domains. As such, most of the mechanisms of CLASS pertain to domains, with systems merely being a special case thereof. For example, whereas one might talk about the lifecycle of a system, one can also discuss the lifecycle of a domain of knowledge.

Domains can be thought of in a set-like space where domains may subsume one another, partially overlap, or depend on one another's content. We do not define a stricter topology for the interrelation of domains at this time, as the best understanding of relationship between domains will likely take time to emerge and might even be fluid.

## 2.2 CLASS as Activities and Monitoring

CLASS operates on the basis of core assurance activities on domains. This is shown in Fig. 3, which is a refined view of the action/monitoring loop of Fig. 2. The middle of the figure depicts a single domain changing over time. Concrete artifacts of a domain are assured (a form of retrenchment over assuring the abstract domain). In the figure, the concrete artifacts of a domain (of types currently anticipated by CLASS) are shown in the left half of the domain. Assurance arguments are written to assure the properties of these artifacts, and processes are enforced in order to aid in assuring and maintaining assurance of the domain artifacts. These are shown in the right side of the domain.

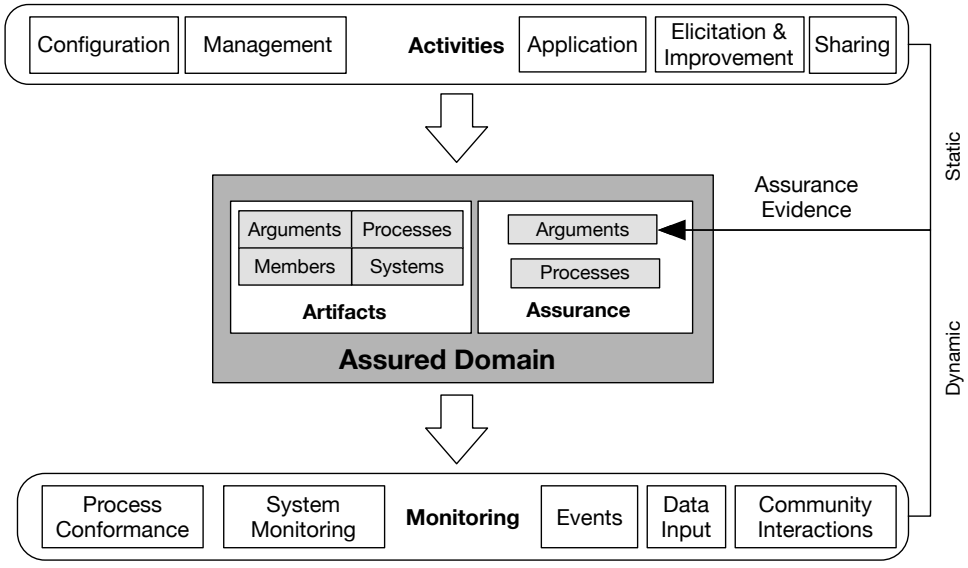


Fig. 3 Abstract view of CLASS functions operating on a domain over time.

This figure depicts the kinds of domain artifacts, activities, and monitoring approaches recognized in the present CLASS design and implementation. The following subsections describe these elements in more detail.

  

## 2.3 Domain Artifacts

CLASS is designed to support a domain's information artifacts in whatever form they might appear. However, there is a set of particular artifacts that are considered primary elements in CLASS.

Primary domain artifacts are:

- **Process**: A process is a partially sequenced set of actions conformant with the Business Processing and Modeling Notation 2. A process is a workflow. It is similar to a flowchart and can be automatically executed within CLASS. The role of processes in domain is to capture expert knowledge within a domain about how things should be done. This includes ordering, appropriate roles, and conditional action. Furthermore, processes that are automated can be logged and monitored. This provides evidence to assurance arguments.
- **Arguments**: Argument is the primary means of assurance in CLASS. Arguments are inductive logical narratives that compute belief, of one or more domain members, in why a particular property is true. The property and the subsequent argument must be rigorously stated. CLASS does not constrain arguments to a particular language or notation. However, it provides immediate support for textual narratives and GSN-based arguments (Goal Structured Notation) [cite]. In the CLASS model, arguments also include evidence that can be directly tied to the claims of an argument in support of those claims.
- **Members**: Members are the human participants in a domain. The primary roles within a domain are administrator, author, and user. An administrator oversees management of the domain within CLASS. Users use the artifacts of a domain within other domains. Authors create the artifacts of a domain within CLASS. Many domains will have far more specialized roles conformant with their defined processes and arguments.
- **Systems**: Many domains will construct systems, which are working tools designed to solve real-world problems. Assuring properties of such tools, primarily emphasizing safety, is the original intent and focus of CLASS. Such systems exist across a lifecycle and can be thought of belonging to a domain, or sub-domain of their own, perhaps with other support artifacts.
- **Other Data and Tools**: While CLASS has a set of core tools and information that it applies in domain assurance; it is assumed that domains and their assurance will use many specialized tools and information formats. As such, CLASS is designed to be extensible for integration with domain specific information and tools.

## 2.4 Domain Assurance

A domain is assured by applying both arguments and processes:

- **Processes:** Processes enforce/observe conformance with expert methods of obtaining desired results. They represent expertise in 'how' to obtain given properties for domain artifacts. They also enforce activities whose post-conditions lend themselves to assurance argument. Finally, aberrations from enforcement expectations are monitoring events that can lead to reparative assurance reactions.

  

- **Arguments**: An Assurance argument is a rigorous, inductive logical structure that can be used by a human to calculate 'belief' in an assured property. In CLASS, assurance arguments are constructed to represent belief of domain experts in the properties of their domain artifacts. Domain members can apply them in order to assess their level of agreement (or disagreement). Those external to a domain can interface the arguments as a means of understanding domain artifact value.

## 2.5 Avoiding Confusion over Constructed Artifacts and Applied Artifacts

It is important not to confuse the arguments and processes built as artifacts within a domain, with the argument and processes artifacts utilized by a domain. A domain might build argument and process templates, and then use argument and processes from another domain to assure its own templates.

In general, the left-hand side of the above figure shows artifacts constructed from a domain. The right-hand side shows assurance arguments and processes applied by the domain. While the latter might originate from source material in the current domain, they often will not. In practice, the assurance of artifacts will combine external template resources with internal domain knowledge.

## 2.6 Activities

The activities of CLASS are defined by a matrix of interactions between users, domain artifacts and domain assurance. These actions can be generalized into categories that are largely representative of CLASS features. The resulting categories are useful for conceptual design of CLASS as well as for exposition to the reader. The remainder of this sub-section presents them.

*Management*

Management involves building and maintaining a community of experts within a domain. Management's goals are to define a domain, manage domain participation in the domain, and encourage an assurance-driven culture. Community can take many forms and the rules can differ on a domain-by-domain basis. Thus, the actions taken in management activity are variably applicable, but can include:

- Defining mission statements.
- Vetting proposals.
- Debating on email forums.
- Adding/removing members.
- Promoting/demoting members.
- Advertising a domain.
- Inviting domain consultants and guests (who are not domain expert members) for oversight and advice.
- Interaction with related domains in their activities.

  

*Elicitation*

Elicitation is the rigorous identification and enhancement of a domain's knowledge. CLASS tools are designed to help explicate argument from domain knowledge. Many types of information present in a domain may contain implicit or explicit argument. Examples include:

- GSN arguments
- Forms and checklists
- Processes and plans
- Standards
- Reports
- Lessons learned
- Textbook material
- Oral traditions
- Policies

As CLASS matures, more explication mechanisms will appear for forms of assurance information within domains in the form of elicitation artifacts such as elicitation processes and guidelines. Each elicitation mechanism is modeled in class as a domain in itself (with its own assurances) such that adding new methods is a modular activity.

Currently, CLASS supports three generic assurance elicitation methods:

- *Argument Retrieval*: Many documents and oral traditions contain arguments. These are often embedded in the structure and text of a document or story. CLASS includes, by default, an argument retrieval process to extract argument from existing text documents. This process requires an argument expert to enact the retrieval and interaction with domain experts and document authors.
- *Argument Recording*: A process is provided through which domain experts can directly record arguments as domain artifacts. This process includes interaction with argument experts as a means to access resources to strengthen and refine arguments after they are written down for the first time.
- *Process Conversion*: A well-defined process (such as one stated in BPMN2) can be annotated with the pre-conditions, post-conditions, and invariants of its tasks. The resulting process can be converted into an informal, inductive argument about any final post-conditions or invariants of the process. The techniques applied here are equivalent to static-analysis of programs, but construct an informal inductive logic tree to be interpreted by humans.

More detail about argument retrieval and recording methods are provided in the Domain Arguments paper [1].

Elicitation is also applied to processes. A given domain might have many processes to follow based on expertise within the domain. Using the BPMN2 notational framework, CLASS relies on the community of expertise that surrounds this workflow language. Thus,

  

CLASS applies existing expertise in process elicitation from the business process modeling community. For more information on business process modeling, and how it relates to process elicitation, see the Camunda Engine's training website [2].

In some cases more than one elicitation mechanism can be applied to the same content. Oral traditions and documents can be analyzed to elicit both process and argument. Process can then be assured through further argument creation. For example, a form might contain both a process and an argument about properties resulting from the form's proper completion.

*Improvement*

Any artifact that can be elicited can be improved by continued analysis of the artifact. In particular, CLASS is designed to use three primary mechanisms of assurance knowledge improvement:

- *Community Interaction*: Domain content is designed to be shared and visible between members of a domain community through the CLASS tools. Domain experts can comment on and potentially edit one another's work. Domains can be established with rules to collaborate or compete to build useful artifacts. Both process and argument authoring are supported directly by CLASS for community engagement.
- *Expert Analysis and Feedback*: CLASS models argument experts as existing in advisory and available analysis roles. Thus quality of argument is left to domain management but with the availability of support from argument experts. Expert feedback is integrated with argument and process elicitation procedures. Further improvement is available through argument participation in community interactions.
- *Monitoring*: Response to live monitoring is an important part of artifact improvement. Monitoring is discussed further later on in this section. For each type of monitoring, there are opportunities to improve process and argument artifacts through analysis of monitoring results.

CLASS does not explicitly define a development methodology for continuous improvement. It is assumed that iterative processes dominate CLASS as with other modern methodologies, but this does not preclude waterfall design where it is better suited to the domain. It is assumed that each of the above activities can take place wherever necessary, with the necessary cascading dependent activities, as required for the improvement/refinement methodology applied. Thus, agile, waterfall, the V model and modular composition are all valid under CLASS.

*Configuration*

Recall that CLASS designers anticipated a network of many overlapping and useful domains that can rely on one another's content. In CLASS, configuration determines how a domain applies the domain artifacts of other domains to it's own assurance. This includes:

- *Resource Selection*: Determining what assurance artifacts to import from other domains.
- *Tailoring*: Tailoring of imported assurance artifacts to the particular needs of this domain.

  

- *Responsible Usage*: Responding to any announcements of issues or problems with important resources. Reporting of issues with usage to source domains. Updating imported resources to new versions that resolve issues or improve assurance, utility, etc.

The configuration problem for artifact usage is well understood within the software engineering tooling domain. Thus CLASS relies upon existing technology to implement resource configuration.

*Application*

Application is the use of domain artifacts. Important forms of application that are built into CLASS are:

- *Process Application*: Execution of BPMN2 processes in a workflow engine
- *Argument Template Application*: Applying argument templates to construct new assurance arguments
- *Systems Application*: Operating systems such as tools imported as artifacts from other domains.

Such activities are the vast majority of time spent in system development, but may occupy various amounts of times in more generic knowledge domains depending on the nature of process and organization within a domain.

*Sharing*

Sharing is the process of publishing and supporting domain artifacts with other domains. CLASS supports sharing through

- Bundling knowledge into resource packages.
- Releasing versions of packages.
- Informing users of problems with packages.
- Informing users of new versions of packages.
- Demonstrating and presenting a system's assurance arguments to concerned parties.
- Providing the assurance arguments of a domain's artifacts as evidence and confidence arguments for those domains importing a domain's artifacts.

The last point is a critical component of the CLASS model. By assuring domains, assurance becomes input to the assurance and confidence in usage of other domains. This effect is intended to provide the 'assurance glue' between domains, as more domains rely on one another. For complex systems relying on many domains, the assurance of imported domain artifacts is critical to the successful assurance argument for the system domain.

## 2.7 Monitoring

CLASS monitors all activity associated with an assured domain. In CLASS, monitoring is synonymous with evidence collection. That is, any monitoring data is assumed to be either

  

- *Current Evidence*: evidence to one or more assurance arguments applied to the domain, or
- *Potential Evidence*: potential evidence to be applied to a non-yet-existent argument applied to the domain.

Whenever monitoring is applied, it is assumed that it is or can be coupled to an assurance argument of some kind. For example, a process showing that different reviewers always conduct on time peer reviews can be evidence of compliance with peer review standards for an organization. On the other hand, peer-reviews that are never completed or turned in very late might be evidence of defeat.

In principle, evidence collection can occur as a result of almost any activity performed by CLASS. In practice, CLASS has implemented evidence collection for a key set of built-in resources, and expects users to supply additional extension of CLASS to automate other forms of evidence collection. These are described in the remainder of this section.

*Process Conformance*

All processes that are executed for the assurance of a domain are monitored. A monitored process maintains a record of all of its state. This includes a complete history of the process state. Each task has a state history that includes the task states, when they were entered and exited, what data was input and output from the tasks, and which systems, tools, and human actors were assigned to and interacted with the task.

Process state history can be utilized to determine process conformance as well as performance. Thus, process monitoring determines assurance of processes being carried out, which serves as evidence that the post-conditions of a process have occurred. These in turn contribute to the assurance arguments applied to a domain.

*Events*

CLASS responds to pre-defined events generated by processes, human actors, tools, and systems. Events can represent discrete conditions relevant to a domain. For example, they can represent transitions in domain lifecycle state, errors conditions in processes, environmental events, or system events.

Event monitoring occurs in class through two mechanisms. The first is input of events to the workflow system, which can respond by logging the event and then optionally triggering any process programmed to act on the event. The second means, not currently implemented in CLASS but part of the design, is as input to a complex event processor (CEP). A CEP can deduce more complex events from simpler events, allowing advanced handling conditions such as real-world circumstances impacting a domain's assurance. CEPs are programmed through the declarative definition of event recording rules and responses.

*Data Input*

Both systems and human actors can input discrete data records into CLASS. The current design supports data input through input of data to process tasks. Interaction can take place between humans and encoded workflow, or between automated systems and workflow. Both

  

can utilize protocols such as messing and event systems. In addition CLASS's applied workflow system supports forms. Forms allow users to directly fill out data about a process, as described in form templates defined with the process.

Data input allows values to be input to automate processing, as well as text to be input for human consumption. Bot can be important parts of assurance evidence. For example, maintenance reports can be thought of as a form that must be filled out as part of a process.

*Artifact Monitoring*

Artifact monitoring is a significant component of CLASS.

An important type of artifact to monitor is the system artifact. System artifacts can be monitored in their live state. This is accomplished with sensor applications, byte-code manipulated software, and direct monitoring of designed software output. All three of these techniques can serve as evidence in support of an assurance argument's assumptions about the operational state of a system.

Other artifact types can also be monitored. For example, artifact download by other domains can be logged and observed. Issues reported back to the domain about artifacts are a form of monitored state about the effectiveness of these resources. Such monitored state serves as positive and negative evidence to their associated confidence arguments.

*Community Interactions*

Finally, CLASS supports interaction between people within its online domain communities. This is itself a form of monitoring between participants as they interact on domain artifacts. Communities generally use self-policing and administrative oversight to maintain community norms. In addition, interaction between community members is the basis for collaborative and competitive development of domain artifacts.

The activity of community members is tracked in software typically associated with such services. Logins, participation, messages, and activities are logged. This information can serve, for example, as evidence to the assurance arguments and confidence arguments about a domain community's role in assurance of artifacts. Participation rates, voting results, and degree of consensus are examples of such evidence.

## 2.8 Domain Artifact Lifecycle

A domain artifact has a lifecycle that characterizes the kinds of activities that take place on the artifact over time. Activities during the creation of domain artifacts, for example, might differ from those in the maintenance of a domain artifact. Likewise, a domain itself has a lifecycle that anticipates the extent and distribution of its artifact lifecycles. A domain in its creation phase is more likely to have domain artifacts being created and discarded. A domain in a maintenance lifecycle phase is more likely to have domain artifacts that are being maintained or repurposed.

Currently, CLASS focuses on modeling the lifecycle of the domain, rather than individual artifacts. Future work would likely change this to model artifacts independently.

CLASS has been designed to support programmable lifecycles, using state-of-the-art software project management technology. Therefore domains can model a lifecycle for themselves and their artifacts that is most useful. However, CLASS defines a default lifecycle that

  

otherwise applies to a domain. This section describes the default lifecycle for a domain and describes the activities that take place in each lifecycle phase.

Fig. 4 illustrates the CLASS domain lifecycle. Circles represent the core phases. Milestones are represented by small squares. Edges of the graph represent available lifecycle state transitions. Milestones are not states and are immediately passed-through. (They can be thought of as events.) A domain is always in one and only one lifecycle state until it is terminated.

The remainder of this section explores the default domain lifecycle stages.

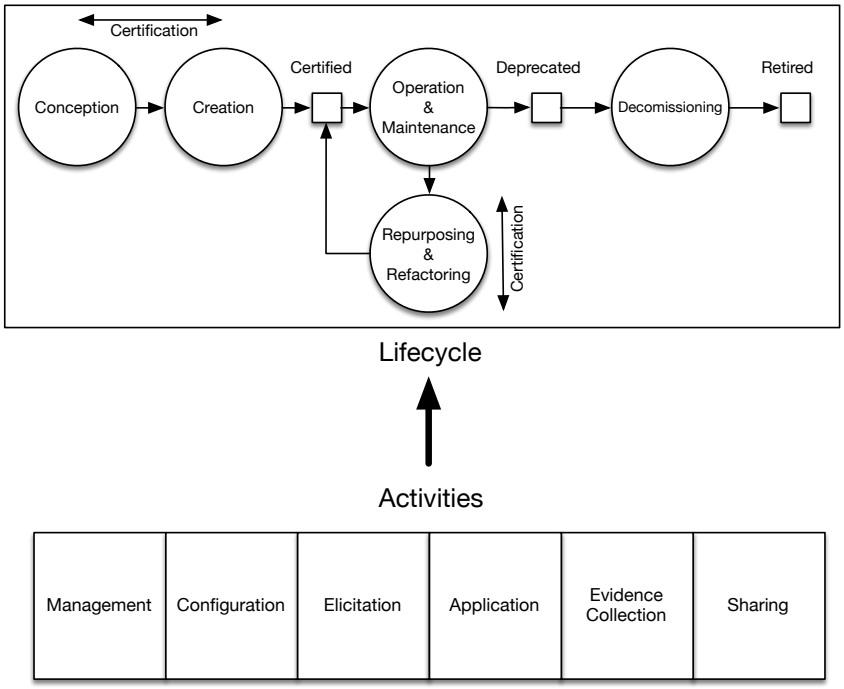


Fig. 4 The CLASS lifecycle and activity interactions.

*Stage: Conception*

The conception lifecycle phase is entered when a realization is made that there is a circumscribed region of knowledge for which artifacts should/could be produced. Some domains are coupled tightly with disciplines and techniques. Others might be more abstract and only emerge later. System domains only emerge when the request for a system emerges. At this stage of a domain's lifecycle, the primary focus is on gaining a better understanding of what the domain would represent. Actions during this phase can generally be mapped as follows:

- *Management*: Naming the domain. Identifying potential representatives for the domain.
- *Explication:* Identifying some of the assurance arguments that are an important part of the domain.
- *Configuration*: Determining what other domains exist around and in this domain. Getting a sense of the domain connections that would surround the concept of the domain. Changing out assurance resources. Updating assurance resource versions.

  

- *Application*: Applying procedures to help define the domain and setting template or sketch arguments in place that will help assure the domain. Arguments might include confidence in the proposed domain's potential artifacts.
- *Sharing*: Announcing the concept of a domain to other domains, and getting their feedback on its utility, efficacy, etc.
- *Evidence Collection*: Determining preliminary evidence associated with the arguments put in place.

*Stage: Creation*

The creation lifecycle phase involves concretely building a domain and its requires assurance knowledge. Activities here include

- *Management*: Assigning personal and their capabilities and roles. Overseeing activities permissions and assignments.
- *Explication*: Analyzing, extracting, and authoring any arguments that are part of the domain.
- *Configuration*: Importing useful and/or relates resources packages from relevant domains. Using these resources during the other activities of creation. Tailoring resources for this domain. (Note, this includes certification arguments).
- *Application*: Following the processes defined during explication and configuration. Following guidelines provided by domains and this domain. Analyzing and iterating domain content to improve and complete it.
- *Sharing*: Packaging up domain content into one or more packages when arguments. Publishing the shared packages so that other domains can use them.
- *Evidence Collection*: Collecting evidence from processes followed, artifacts created, tests run, etc.

*Milestone: Certified*

Certification is a milestone having demonstrated that the domain is sufficiently conformant, in spirit and letter, with applied standards and regulatory requirements. It has taken place throughout conception and creation, and is maintained as necessary in later lifecycle stages. Certification has occurred by creating sufficient argument of support for conformance to standards and regulations.

*Stage: Operation and Maintenance*

Operation is a stage where a domain is actively applied. Systems in the domain may be operating. Other domains might actively apply arguments of the domain. Activities here include

- *Management*: Oversight of the live domain.

  

- *Explication*: Analyzing, extracting, and authoring any defeaters identified against the domain in the field. Updating arguments based on knowledge of the domain obtained during its operation.
- *Configuration*: Determining if resources from other domains are meeting needs. Reconfiguring to choose new resources and needed. Continuing to tailore resources to meet goals.
- *Application*: Following the processes defined during explication and configuration for operation of the domain. Making small updates and releases to resources of the domain, including fixes for arguments and systems.
- *Sharing*: Announcing detected deficiencies to other domains using this domain's packaged resources.
- *Evidence Collection*: Collecting evidence from processes followed, maintenance reports, incident reports. Collecting evidence of invariants on system operations including assumptions and claims of the argument.

*Milestone: Deprecation*

Eventually, any domain might become obsolete. It might be deemed inferior to an alternative, or no longer relevant to the assurance community at large. Its systems might be deprecated. In such cases, a domain is marked as deprecated, to indicate that users are dealing with resources that might not be actively maintained.

*Stage: Repurposing, Refactoring*

Domains are not static and are likely to see updates and changes. When changes to the purpose of a domain are sufficiently large, but the domain has sufficient value, a domain undergoes repurposing and refactoring. Thus, it might be refactored with a more useful relationship to other domains and repurposed with a differing definition of its domain.

- *Management*: Oversight of the transition in definition of the domain.
- *Explication*: Analyzing, extracting, and authoring the differences between the previous domain and the new domain.
- *Configuration*: Understanding the new relationships to other domains. Defining those relationships precisely.
- *Application*: Following the processes defined during explication and configuration for operation of the domain. Making small updates and releases to resources of the domain, including fixes for arguments and systems.
- *Sharing*: Announcing and negotiating changes to the domain between new and older related domains.
- *Evidence Collection*: Collecting evidence from processes followed, and conformance needed to help make the transition.

  

*Decommissioning*

A deprecated domain can be decommissioned. Here, the domain is solidified, so that it no longer changes. Active management ceases and the resources of the domain are maintained as artifacts for the greater understanding of the community. Activities during the decommissioning lifecycle phase include:

- *Management*: Oversight of the transition in definition of the domain.
- *Explication:* Arguing for decommissioning to occur in a safe manner. Using arguments from configuration as well as any new arguments now apparent. Using decommissioning documents as a source of safe decommissioning domain argument.
- *Application*: Following the processes defined for decommissioning. Applying guidance for decommissioning.
- *Sharing*: Negotiating decommissioning with related domains.
- *Evidence Collection*: Collecting evidence from processes followed, and conformance evidence for standards and guidance. Observation of invariants for systems in the decommissioned stage that are assumptions and claims of decommissioning arguments.

*Milestone: Terminated*

A terminated domain is in a final state. Decommissioned domains that require active monitoring are not in the terminated state until evidence collection is no longer required. For some domains, there is no timetable for termination state.

# 3 Server Toolset Implementation

The CLASS Server Toolset has been implemented, experimentally, as a prototype system derived directly from the high-level design.

## 3.1 Implementation Philosophy / Soft Requirements

Goals of CLASS implementation that were not part of the conceptual design include:

- *Re-use of the state-of-the-art*: There are many advanced project management systems that relate to the goals of CLASS. While not focused on assurance, such systems and tooling have many features of direct utility.
- *Integration-based design*: Modern software systems are frequently not written 'from-scratch', but instead assembled as a collection of interacting, pre-defined services. Integration of services and components, typical of enterprise software design, is the approach with CLASS where possible.
- *Open software service philosophy*: The most successful management systems tend to have collections of tools that enable communities to establish work regimes with their own rules and conventions [cite]. CLASS applies this philosophy in choosing services that do not impose rigid processes.

  

## 3.2 Logical Architecture

*Component View*

Class consists of three key component types: server, client, and repository. These interact as shown in Fig. 5. CLASS Servers host domains. Each server can host multiple domains, and there can be an unlimited number of servers. Clients participate in domains. Servers publish domain artifacts to repositories. They also import artifacts from repositories for use in assuring their domains.

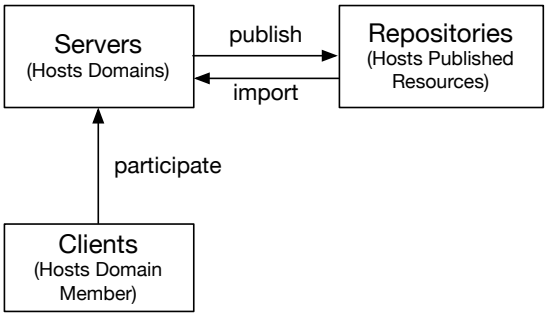


Fig. 5 The basic components of the CLASS service.

*Server*

The logical view of a CLASS server is shown in Fig. 6. A CLASS server hosts one or more domains. Each domain contains artifacts and their assurance as data. These are supported through services for the domain, including a wiki to host a community and authoring of artifacts, a workflow system to enforce process, a configuration application in which the domain can import artifacts from other domains (to apply to this domain's assurance) and a means to publish artifacts to repositories, keeping track of to where and when publication occurred.

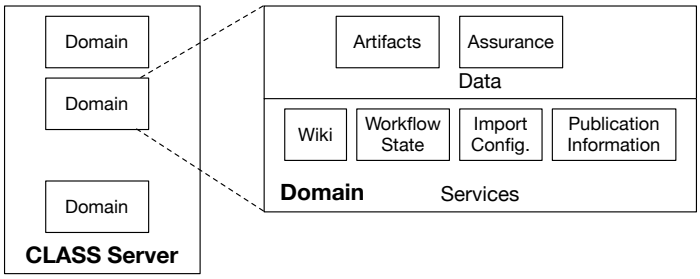


Fig. 6 A logical view of a CLASS Server

*Client*

A client interacts with a server as shown in A logical view of artifacts is necessary if perspective of interacting with a domain is shown in Fig. 7. The client has two key ways of interacting with a domain.

  

- **Web View**: The web view consists of interaction with a domain through a domain service. As depicted in the server diagram, the server hosts a wiki, process engine, and git repository for each domain. Each has a web client front end accessible through clients via a web browser. Through the wiki, the client can:
  - *View wiki content* that the user has permission to see.
  - *Write wiki text* where the user has permission to author.
  - *Edit arguments as text* where the user has permission. Design for GSN editing in the wiki page have been drawn but not implemented.
  - *Comment on others work*.

  In addition, administrators can use applications embedded in the wiki in order to:
  - Create a new domain.
  - Configure a domain with imported domain artifact resources.
  - Publish domain artifacts.

  The web client can also access a task interface to the process engine. Here, users can:
  - Perform tasks assigned to them.
  - Fill out forms in completion of tasks.
  - Observe processes underway.
  - Start and stop processes (with proper permissions in the domain).

  The web client also gives users access to a visual Git server where clients can observe the file-form of any artifacts associated with the domain. This includes arguments, processes, and documents defined in the domain. Users can use this interface to
  - View resources including individual files.
  - Pull files to the local client for editing.
  - Commit files back to the repository.

  

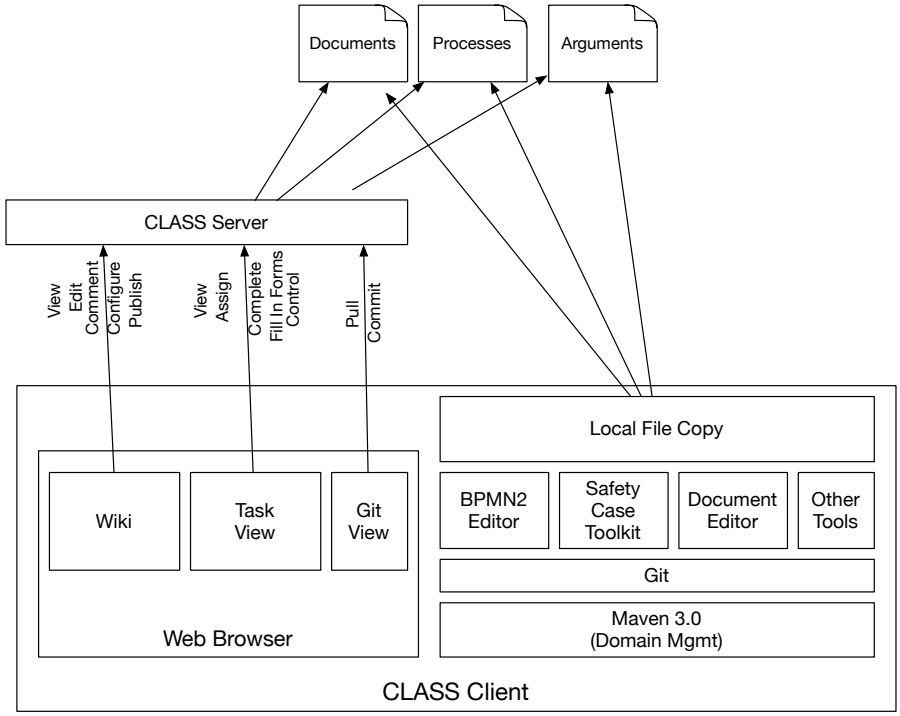


Fig. 7 Client interaction with server and artifacts.

- **File View**: Resources pulled to a local client through the GIT system are maintained as a local view on the client using the typical GIT model. From here, assets can be edited through local BPMN2 process editors, GSN editors such as the Safety Case Toolkit (from Dependable Computing) and document editors. Other tools associated with a domain can be part of the file repository and accessed through this mechanism.

There is some contention between the Web Service mechanisms and the Git file-based mechanisms applied by CLASS. The file-based mechanisms are more project oriented and similar in organization to the way source code projects are shared between distributed team members. The Wiki interface approach to editing is designed as an experimental approach towards encouraging collaboration and community in domain authoring. It is not clear at this point which approach to interfacing CLASS is preferred. Both are prototyped for now.

*Repository*

A logical view of a repository is shown in Fig. 8. This figure also shows how a server interacts with the repository.

  

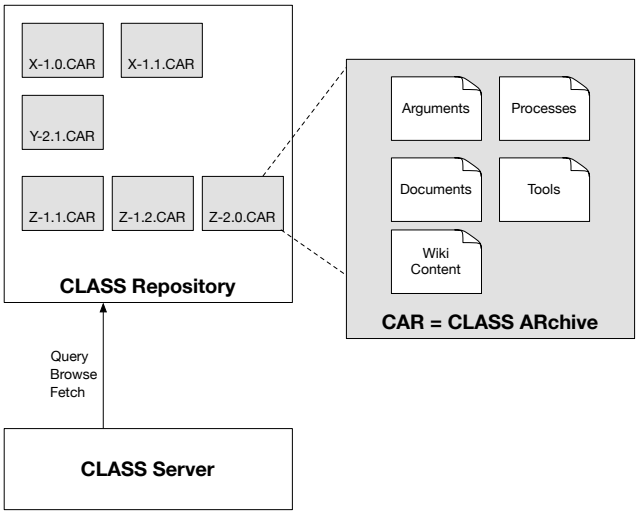


Fig. 8 Logical view of repository

A repository contains CAR files. This is a CLASS Archive. A CAR file is similar to a ZIP file and contains arguments documents, processes, tools, wiki content, and other artifact types defined in CLASS. It is defined in a maven format so that CLASS can automatically import the CAR file and its resources from a repository into a domain. CLASS then automatically distributes the resources into their appropriate services and containers within the CLASS server, such as the wiki and process engine.

CLASS servers interact with the repository by querying it or browsing to view available CAR files and their versions. The CLASS server performs fetching CAR files automatically whenever a domain configuration is changed to use new resources.

## 3.3 Service Architecture

The service architecture of CLASS is a stack depicted in Fig. 9. This stack supports both the client and server, where the each stack layer is replaced with the client or server view/role for the client and server, respectively. Those elements in gray are currently implemented in the prototype. Those elements in white are designed but not prototyped.

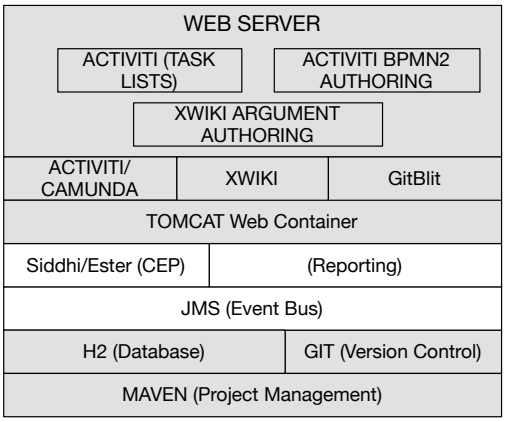


Fig. 9 CLASS Service Architecture

  

At the lowest level of the service stack is Maven 3 [6]. Maven is a software project management solution. Most importantly, Maven is responsible for managing a domain over its lifecycle at the command-line, file system level. CLASS models a domain as a form of maven project. It models CLASS artifacts as project artifacts, and published CLASS assets (CAR files) as maven resources.

Maven support s features such as declarative project configuration, dependency resolution and automatic project resource fetching and installation. As such, the low-level configuration and lifecycle management of a CLASS domain are almost entirely implemented through the project model of maven.

Where standard maven models do no suffice, CLASS applies maven's configuration languages to more accurately model domain semantics. The CLASS domain lifecycle, resource file formats, domain, and artifact project types, are all declaratively defined and applied to Maven.

Maven is also responsible for bootstrapping a CLASS server. From a single file, maven downloads and sets up all of the enterprise components necessary to run a CLASS server, including databases, web servlets, and web applications. Management of the CLASS server (hosting domains) is conducted through maven commands.

Maven installs an H2 relational database engine [8]. The database holds all elements of a domain's state that are not stored directly as files. This mostly serves the web services hosted by the CLASS server. The H2 Database is a common open source relational database system. It supports in memory, and in file system modes, so is valuable for prototyping and moderate release workloads. As most CLASS activities are low frequency, low content I/O operations, it is expected to scale with anticipated CLASS workload.

Maven installs Git [9], on a class server. Git and Maven must also be installed on any client working on domain artifacts. Git is responsible for version control of file based assets, much in the same as the database is responsible for relational assets. Both can be thought up of as the information repository for a domain.

The event bus, reporting, and complex event processing layers of the service stack are intended to support advanced monitoring of complex event sequences, and advanced reporting of live conditions over a domain and its artifacts. Their role in monitoring for the high-level CLASS design was discussed in previous sections. At the end of the CLASS resource effort, these design elements had not yet been prototyped, and technology selection was still pending. These would be particularly valuable for interaction with a live system, and could in practice be replaced with professional grade software system management software available from a commercial entity.

A Tomcat engine [5] sits at the next service layer. Tomcat is a web service container. It can support many applications over the Internet through plug-in service definitions. Three such services are applied in CLASS.

One service running on Tomcat is a workflow-processing engine. The Activiti [3] or Camunda [4] workflow service can be utilized with CLASS. (Both are forks of the Activiti project.) The current prototype utilizes Activiti, although future plans are to shift to the Camunda for its currently superior web front-end technology. The workflow engine is responsible for running processes on behalf of a domain. The high-level purpose of the workflow engine was discussed in previous sections.

Tomcat also hosts the XWiki platform [7]. XWiki is a second-generation wiki platform. It can host user-defined content that includes simple programs. In this way, the wiki serves as

  

both a content collaboration platform for domains, as well as a front end for the creation, configuration, and publishing of domains.

Finally, Tomcat hosts GitBlit [10], an easy to use but reliable web-based front end for the GIT file control system described above. This service allows clients to easily scan and fetch, in a version-controlled manner, the artifacts associated with a domain, should interaction directly with the wiki not be desired or preferred.

Within the above applications, the server stack supports process editing (in Camunda) task list management for users participating in workflow processes, and authoring of arguments (as text in the wiki.) Separately, and not included in the figure, standalone editor tools, such as BPMN2 editors and the Safety Case Toolkit editor, can be used to edit domain artifacts.

There is some remaining contention, as might be expected, between the project/file view of CLASS and the wiki/content view of CLASS. This contention remains and while navigable within the CLASS design, would need further resolution in maturing tools. The former provides a project-oriented view of a domain. The latter provides a collaboration space-oriented view of a domain. The two models remain as their relative merits are still being assessed.

### 3.4 Dataflow

A dataflow view of a CLASS is presented in Fig. 10. This view abstracts away the client/server distinction. In this view, wiki and authoring tools handle document, argument, process, system, and domain configuration authoring. Domain members participate in this authoring.

Document files of various formats, GSN argument files, BPMN2 process files, XAR wiki files, and POM maven files represent the resulting artifacts. Maven manages these files. Maven packages them into CAR files for publication to repositories. It also handles unpackaging the repositories imported from repositories, installing the various component files in the requisite tools on the server.

Events are generated and received both at the XWiki level and the Maven level of CLASS abstraction. The workflow system is currently the main broker of events in the CLASS prototype implementation. As discussed earlier, event bus transport and complex event processing are not currently implemented.

  

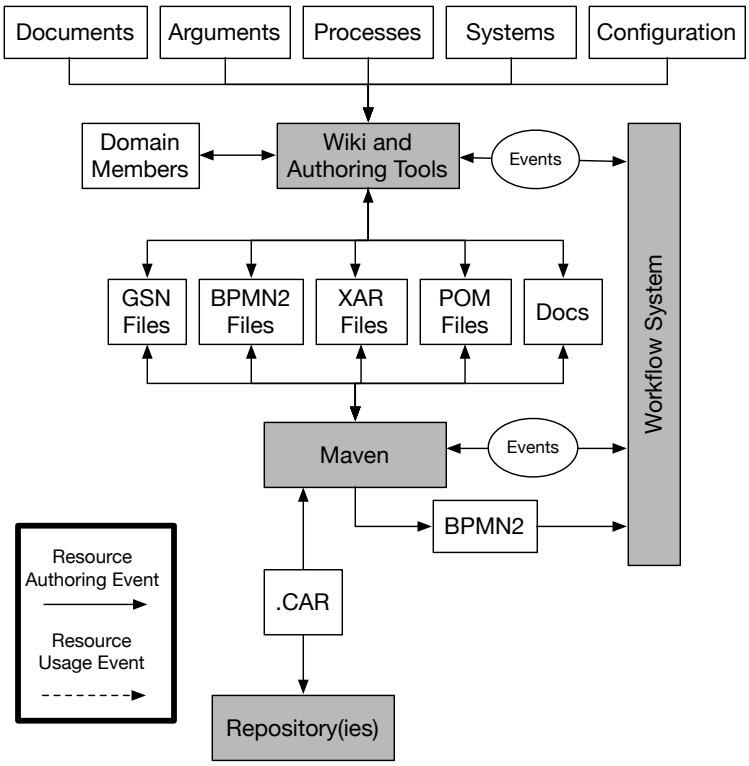


Fig. 10 CLASS dataflow architecture

# 4 Default Artifact Packages

CLASS operates on the basis of importing domain knowledge into new domains in order to support their assurance. Therefore, a base set of assurance capabilities are presented in core import packages. These design domains of knowledge directly intended to support CLASS. In addition, packages created during empirical studies have been created and are presented as example resources. The collection of pre-existing CLASS archives, from their respective domains, are presented in Fig. 11.

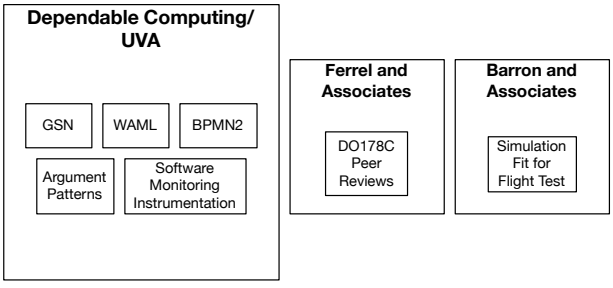


Fig. 11 Current resource packages

For the purpose of not overriding potential domain communities, all of the default resources haves been defined in domains named for institutions, rather than conceptual bounds.

  

The core resources of CLASS are presented at the left of Fig. 11. The packages supported will be:

- **GSN**: A package including references to GSN materials and the Safety Case Toolkit-Light editor, in which GSN can be authored.
- **WAML**: A package with a guide to workflow annotation and markup, as described in the domain arguments paper. This package also includes a BPMN2 file for the elicitation processes of argument recovery and argument recording present in that work.
- **BPMN2:** A package with references to tutorials, tools, and online resources for editing Businesses Process and Modeling Notation 2 processes. This includes a guide to making these processes executable within the CLASS environment.
- **Argument Patterns:** A collection of standard argument patterns in use at dependable Computing. There might be multiple libraries supported if CLASS should be successful, such as security case templates, and systems of systems templates.
- **Software Monitoring Instrumentation:** A package that can contain tooling and information about instrumenting software checking of argument constraints and assumptions.

Experiment result packages will include an argument for DO178 peer review conformance, in an early draft form, a fit for flight test argument for the NextCAS system simulation defined in the experiments on CLASS.

# 5 Implementation Status

The CLASS Server Toolset is a functioning prototype, although it has only been implemented to the extent necessary to determine its feasibility and the nature of component potential. The following components are functional:

- CLASS Server Self-bootstrap and setup.
- Domain creation through Maven and/or Wiki.
- Domain configuration through Maven and/or Wiki.
- Domain text authoring in Wiki.
- Argument authoring in text in wiki.
- Argument authoring in text or GSN at file level.
- Process authoring through standalone tools at file level.
- Packaging and publishing of domain resources as CARS at Maven level.
- Orchestration of human activities through a workflow process engine.
- Forms and data content in processes.

The following features are designed and partially implemented:

- Publishing of a domain's artifacts to a repository via the Wiki.
- Automatic storage and injection of Wiki content in CAR files.

  

- Automated software-to-software automation through workflow processes, e.g., directly from Wiki action to Maven action through workflow triggers.

The following features are designed but are not implemented:

- Complex event processing engine integration into CLASS for observing event patterns.
- Integration of reporting infrastructure into CLASS.
- Integration of software artifact modification via Zephyr correlated with argument constraints and assumptions.

To summarize, the majority of effort performed in implementing the CLASS Server Toolset focused on modeling what information needed to be obtained and how it needed to be assured. Further effort focused on the nature of how assurance knowledge could itself be made relevant to applied engineering where the application of domain-agnostic template libraries was falling short of quality considerations.

The result of a focus on domain-oriented arguments shifted focus away from active monitoring. Therefore, this work did not take place at the tooling and prototype implementation level. Instead, it remained compartmentalized in the research results published elsewhere in this report, subject to unification only through high-level design as presented earlier in this report.

# 6 Summary

The CLASS Server Toolset has been designed and implemented to demonstrate a domain knowledge-centric approach to the development of assurance. It relies heavily on assurance elicitation, within expert domains, and the application of the result to system domains.

The current CLASS tools are sufficient to demonstrate the core vision of domain collaboration in building and assuring domain knowledge. Certification is also supported through import of certification and conformance domain resources, such as the peer review argumentation presented elsewhere [1].

However, the above is only half of the expected lifecycle of a given domain. What remains to be determined is how more active monitoring, in particular through active processes defined as domain knowledge, can be applied to the complete lifecycle of a system or knowledge domain.

# 7 References

1. Jonathan Rowanhill and John C. Knight, *Domain Arguments in Safety Critical Software Development*, submitted to 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, Canada (October 2016)
2. “BPMN 2.0 Tutorial: Get Started with Process Modeling using BPMN” https://camunda.org/bpmn/tutorial/
3. “Activiti”, http://activiti.org/
4. “Camunda”, https://camunda.org/
5. “Apache Tomcat”, http://tomcat.apache.org/

  

6. “Apache Maven Project”, https://maven.apache.org/
7. “XWiki: Open Source Wiki and Content-Oriented Application Platform”, http://www.xwiki.org/xwiki/bin/view/Main/WebHome
8. “H2 Database Engine”, http://www.h2database.com/html/main.html
9. “Git”, https://git-scm.com/
10. “Gitblit”, http://gitblit.com/

  