ACE: Age Calculation Engine—A Design Environment for Cosmogenic Dating Techniques

Ken Anderson, Elizabeth Bradley, Marek Zreda*, Laura Rassbach, Chris Zweck*, Evan Sheehan

Department of Computer Science
University of Colorado, Boulder
Boulder CO 80309-0430

*Hydrology and Water Resources
The University of Arizona
Tucson, AZ 85721

Abstract

ACE (Age Calculation Engine; previously called iCronus) is a design environment for analyzing data used in cosmogenic dating methods. It is supported by a software architecture designed with flexibility, scalability, security and safety in mind. These properties have allowed us to create an environment that directly supports the tasks that geoscientists perform as they work on developing new algorithms for cosmogenic dating, such as running calibrations, defining new experiments, and evaluating the impacts of scaling factors on the calculated ages of samples. Our goal is to provide geoscientists using cosmogenic dating methods with a flexible and powerful software infrastructure upon which to base their future research efforts. In this paper, we describe the design of the ACE system and compare it to existing cosmogenic dating software. We also discuss how our system has been evaluated and our plans for future work.

1. Introduction

For the past four years a team of computer scientists (Anderson, Bradley, Rassbach and Sheehan) and geoscientists (Zreda and Zweck) have been working together to revolutionize the way software is used to support the study of cosmogenic dating techniques. We (the computer scientists) are developing the ACE design environment to allow our collaborators and other geoscientists to calibrate cosmogenic methods, calculate ages based on inventories of cosmogenic nuclides, and create, edit, run, and evaluate new cosmogenic dating algorithms.

Cosmogenic dating techniques use the accumulation of certain rare isotopes produced by cosmic rays in materials exposed at the surface of the Earth. The intensity of cosmic radiation is high enough to penetrate the atmosphere and interact with nuclei of atoms within rocks. Production rates are very low; typically tens to hundreds of atoms are generated by cosmic-ray interactions per gram of sample each year. However, on a geological time scale, the accumulation of these nuclides is large enough to be measured in the laboratory. When the concentration of a cosmogenic nuclide has been measured, and the production rates are known, the exposure age of the sample can be determined. This technique is ideal for dating surface features such as meteor impact sites, earthquake ruptures, lava flows, alluvial fans, terraces and landforms associated with the retreat of glaciers.

Existing software in this area consists largely of Excel spreadsheets cobbled together by geoscientists to support their work in understanding the samples they collected in the field. While these tools have their place and can provide a certain amount of utility, mostly to their authors, they come with a host of problems. Assumptions are often hardwired into the structure of the spreadsheet that run counter to the views of other research groups and furthermore are difficult to change or adapt to some other context. These spreadsheets often have a single dating algorithm programmed into them and the structure of that algorithm is difficult to change. Finally, these algorithms often focus on a particular nuclide and expect parameters associated with just that nuclide. If a group would like to use the same algorithm but focus on a different nuclide, they are faced with the prospect of starting with a new spreadsheet, copying bits and pieces of the previous spreadsheet that can be reused, and then implementing new code to handle the new nuclide and its associated input parameters.

Prior to the release of ACE (aka the Age Calculation Engine), two recently developed software programs were available to date cosmogenic nuclides. The CRONUS-Earth series of calculators (`hess.ess.washington.edu/math`) compute sample ages of ^{10}Be and ^{26}Al using an online web server operating as a front end for a set of MATLAB routines. The MATLAB routines are publicly

available but as MATLAB is commercial software the web interface was developed to bypass ownership issues. A perceived problem with this configuration is that Internet access is always required, and the user has limited ability to store results locally or modify the code to examine assumptions used in the dating procedure. In addition, geoscientists must be willing to submit information about samples relating to unpublished research to a web service that is managed by an unaffiliated research group. This constraint can make users uncomfortable as there is no way they can verify that their unpublished data is not being stored indefinitely on a server outside their control. However the CRONUS-Earth calculators do provide a simple, repeatable and internally consistent method in which to date samples.

Cosmocalc (`cosmocalc.googlepages.com`) was developed in Excel to be downloaded locally and modifiable so that Internet access is not required. A modular design was used so that the effects of changes to the methodology can be easily examined. However the choice of Excel limits the number of potential users and Excel lacks comprehensive mathematical and statistical libraries, making development difficult.

In order to address these issues and provide more powerful and flexible cosmogenic dating software to geoscientists, we decided to

1. apply modern software engineering techniques to the design of software in this area (as opposed to hacking equations in spreadsheets),
2. create an environment that made no assumptions about the types of nuclides, scaling factors, algorithms, etc. used by a research group and instead allow these elements to be plugged into the environment as needed,
3. design an architecture for the design environment that enables maximum flexibility, scalability, security, and safety for its users.

But creating a new cosmogenic dating application is simply not enough. Researchers need a design environment that lets them explore "what if" scenarios, creating variations of existing algorithms that might lead to more accurate dating. Such an environment encourages innovation in developing new approaches to cosmogenic nuclide dating rather than boxing in progress with outdated assumptions, fixed sets of constraints, and a single calibration/dating algorithm.

The ACE design environment allows the creation of new dating algorithms that can support a variety of nuclides and scaling factors. It provides geoscientists with a variety of software components that can be "wired together" in a drag-and-drop environment to produce new dating algorithms. Geoscientists can also upload new components into the design environment to make new capabilities available and to allow their research to move in new directions. Our design environment also supports the ability to import data sets that may be needed to support the calculations of the software components. We discuss the design environment in detail in Section 2.

The design environment is supported by a software architecture designed to be flexible, scalable, secure and safe. The ACE architecture is flexible with respect to the components and data sets a geoscientist can use to develop new algorithms. It is scalable in that it can support a single user working in isolation on a single machine while also supporting shared work performed by small research groups. In addition, it can be configured to allow research groups to share published algorithms and data sets with their larger research community. With respect to security, ACE supports multiple types of users with varying access rights. Basic users can load samples and run experiments while more advanced users can define, e.g., new algorithms, components, and scaling factors that are then made available to all users. In addition, ACE takes steps to keep its users' data safe. All data is stored in a database on a user's personal computer and only the owner of that data has access to it. In addition, the information in the database can easily be exported in a number of formats to facilitate data replication and enable archival storage. We discuss the design of the ACE software architecture in detail in Section 3.

In addition to providing a whole new approach to the design and construction of cosmogenic dating software, we are creating artificial intelligence (AI) tools that will integrate advanced analysis services into the ACE design environment. It is our goal to not only meet the research needs of our users, but to actually augment their abilities by automating some of the more-routine parts of the analysis process. These facilities will also help train graduate students in the geosciences with the skills required to perform work on cosmogenic dating. To meet these goals, we are building features into the design environment that can analyze a set of samples automatically and reason from the results to make suggestions about what geological processes may have been at work, whether or not to gather more data, etc. To produce these suggestions, the AI subsystem uses an argumentation logic [9] that captures how expert geoscientists reason about cosmogenic data. However, the focus of this paper is on the software architecture of our design environment and the important properties it provides. Readers interested in our design environment's AI capabilities should see [9] for more details.

The work in this paper is important as it serves as an example of how decisions at the software architecture level can have impact on the flexibility of a software development process (in terms of responding to change requests) as well as on the flexibility and extensibility of the software prototype produced by that process. These characteristics are especially important when engaging in the development

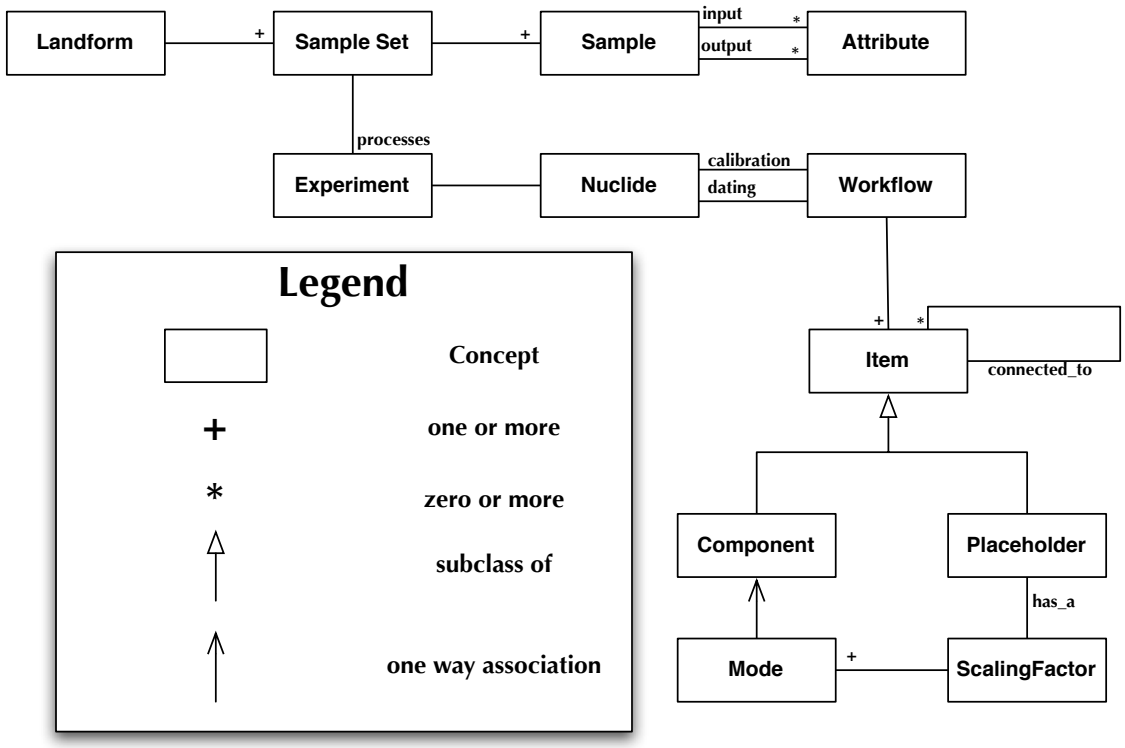

Figure 1. The ACE Conceptual Framework

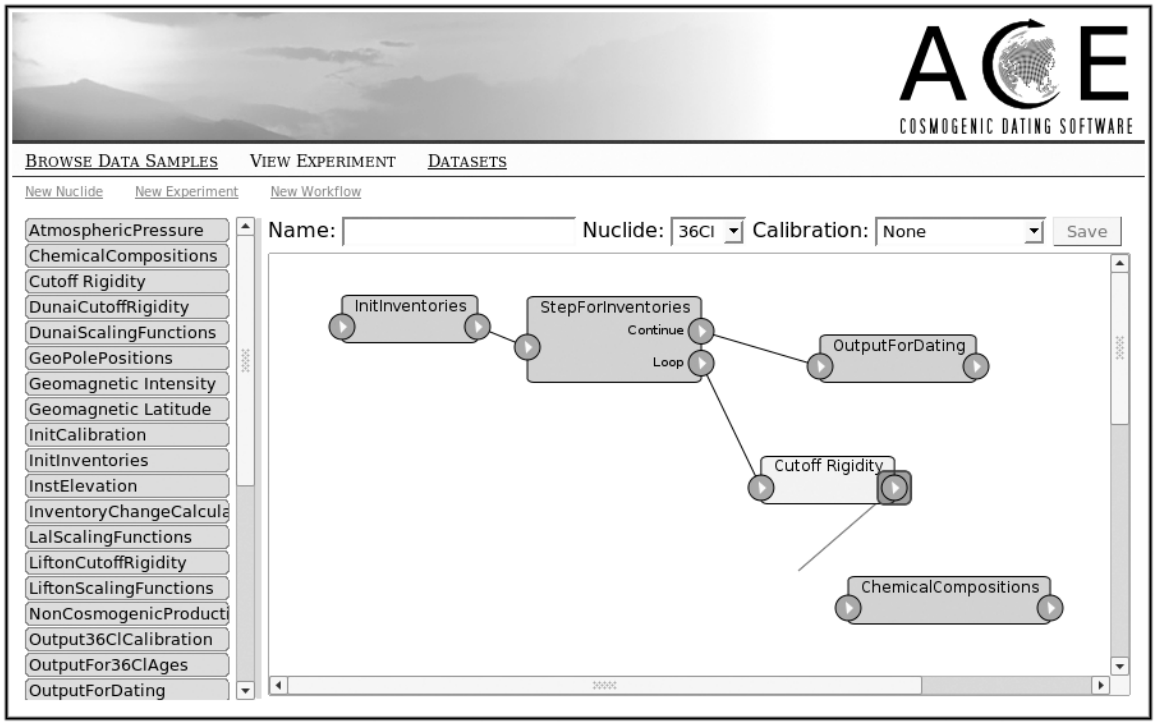

Figure 2. The ACE Workflow Editor

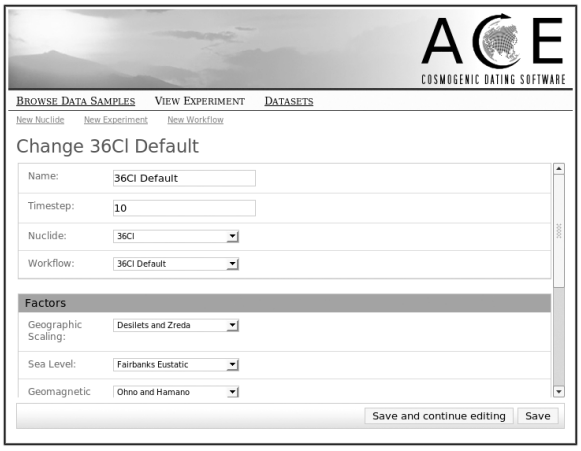

Figure 3. The ACE Experiment Editor

of domain-specific software architectures, design environments, and tools.

This paper is organized as follows. In Section 2, we discuss the conceptual framework underlying the ACE design environment and present details on how it is implemented. Then, in Section 3, we discuss the ACE software architecture and how it provides important properties to the design environment. In Section 4, we discuss ways in which we have evaluated our design environment to verify that it meets the needs of its users. In Section 5, we present related work and then conclude in Section 6 with a discussion of our future research plans.

2. ACE Design Environment

In this section, we discuss the framework that provides the conceptual foundation of our design environment. We then describe the services that our design environment provides and explain how we have implemented both the framework and the services in a proof-of-concept prototype.

2.1. ACE Conceptual Framework

Our design environment is based on the ACE conceptual framework (Fig. 1). *Geoscientists* use *cosmogenic dating methods* to understand the evolution of a *landform*. Each landform can produce one or more *sample sets*. Each sample set contains one or more *samples* (rocks). A sample can contain one or more *nuclides* and has a set of *input attributes* and a set of *output attributes*. Input attributes are either *nuclide-independent* (present for all samples) or *nuclide-dependent* (present only for samples that contain a particular nuclide). Output attributes are calculated by *components* (see below).

Samples are processed by *workflows* that consist of one or more *components* and/or *component placeholders*. Each component consists of a set of *input ports* and a set of *output ports* and performs a specific computation. Components can be connected together via their ports to form workflows with sequential, branching, and looping paths. A component is activated whenever a sample set arrives on one of its input ports. The component will iterate over each of the samples in the set and perform its calculation, storing the result as one or more output attributes on the sample. Once a component has finished processing a sample, it will place it within a sample set associated with one of its output ports. The component will then pass these output sets to the components connected to its output ports, activating them in turn.

A component placeholder is a location in a workflow associated with a particular *scaling factor*. A scaling factor is a calculation that can influence the results of a calibration or dating workflow. For instance, since the earth's sea level has changed over time, a dating workflow that takes sea level changes into consideration will produce different exposure ages than a dating workflow that does not. Each scaling factor can have multiple *modes* in which a different algorithm or a different data set is used to perform its calculation. A different component is associated with each mode of a scaling factor and that component is plugged into a workflow at the location of its associated component placeholder. Indeed, a workflow cannot be executed until a component has been plugged into each of its component placeholders, which corresponds to selecting a mode for each of the workflow's scaling factors.

An *experiment* has a set of input parameters and an associated nuclide. Each nuclide has an associated *calibration workflow* and *dating workflow*. These two workflows must both have the same set of scaling factors. Indeed the only difference between these workflows is that a calibration workflow is used to calculate a set of *production rates* that are used by the dating workflow to calculate exposure ages of samples. After an experiment's nuclide has been selected, a user must select a mode for each of the scaling factors that appears in the nuclide's two workflows. At that point, the calibration workflow can be applied to a *calibration data set* to produce the production rates needed by the dating workflow. A calibration data set is simply a sample set whose samples have been dated using another dating technique (such as carbon-14 dating). Once the production rates have been calculated, the experiment's dating workflow can then be used to process any sample set contained in the design environment.

2.2. ACE Services

Given the conceptual framework described above, we identified the following services that should be provided by the ACE design environment. The current ACE prototype

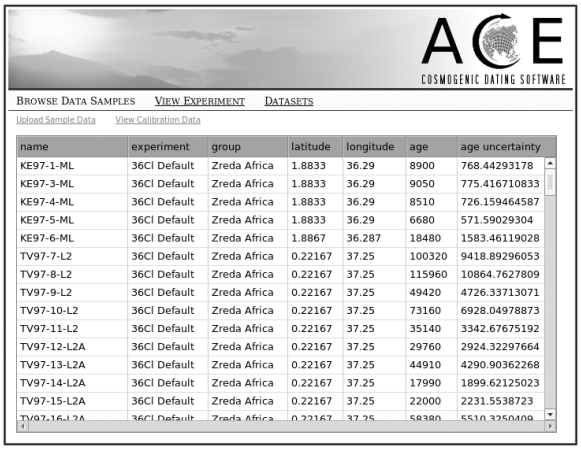

Figure 4. The ACE Sample Editor

implements most of these services, but in some cases, the "logical" functionality discussed here has been combined in various ways so as to reduce the number of "actual" editors/services presented to the design environment's users.

Data Collection Editor: The data collection editor allows users to enter data sets that may be required by components for their calculations. Examples of such data sets include sunspot data, geomagnetic pole positions, sea level data, magnetic field intensities, etc. Using the data collection editor, a user can define the fields that appear in each data set, import data for a particular data collection from comma-separated input files, and edit individual values of a data collection.

Nuclide Editor: The nuclide editor allows a user to define a new nuclide and assign calibration and dating workflows to it. This editor can also be used to specify the nuclide-dependent input attributes that a nuclide expects a sample to have in order to be processed by its associated workflows. In addition, this editor can be used to specify the output attributes that a nuclide's workflows add to the samples they process. This information allows other tools in the design environment to distinguish between "interesting" output values and intermediate values that are calculated by the components along the way to determining a sample's exposure age.

Component Editor: The component editor allows a user to create and edit the components that appear in workflows. Users can write fragments of code using the Python scripting language that specify how a component processes the samples sent to it. The user is also able to specify the data collections required by a component. The component editor uses this information to produce an actual component that can be placed in a workflow and invoked when needed.

Scaling Factor Editor: The scaling factor editor allows a user to create a new scaling factor and define its various modes. Each mode can be given a specific name and the user can specify a component that performs the calculations associated with that mode. Once a scaling factor has been defined, its corresponding component placeholder becomes available in the workflow editor.

Workflow Editor: The workflow editor provides a graphical environment for creating and editing workflows (Fig. 2). Each component created by the component editor and each component placeholder created by the scaling factor editor is available in a palette and can be positioned via "drag-and-drop." Once these objects are in position, a user can connect the ports of components and placeholders to each other using point and click. Once a workflow has been created, it becomes available in the nuclide editor and can then be associated with a particular nuclide.

Experiment Editor: The experiment editor allows a user to create and edit experiments (Fig. 3). It provides an interface to let a user pick a nuclide for the experiment, supply values for an experiment's input parameters and select the modes for each of the scaling factors that appear in the experiment's workflows (as determined by the experiment's nuclide). Once an experiment has been configured, the editor allows the user to select a calibration data set and invoke the experiment's calibration workflow which then calculates the experiment's production rates. The experiment can now be applied to any sample set that contains samples with the experiment's nuclide.

Sample Editor: The sample editor can be used to browse and edit individual samples and sample sets (Fig. 4). It can be used to create new sample sets and offers batch import of sample data from comma-separated input files. (Members of our team of geoscientists tend to collect data about samples in Excel spreadsheets while in the field. It is then trivial for Excel to export comma-separated data files from these spreadsheets for import into the ACE design environment.) The sample editor can also be used to apply an experiment to a sample set and view its results.

Plotting Services: ACE can generate a number of plots to help its users understand the data produced by ACE experiments. When viewing samples in the sample editor, it is possible to ask ACE to plot the ages of each sample with error bars. ACE can also plot the values of one attribute against a second attribute for a set of samples. For instance, our collaborators like to plot the calculated ages of a set of samples against the elevations of those same samples. This plot can provide insight into how the landform associated with the samples evolved over time.

Analysis Services: As mentioned in the introduction, we are also developing services that use artificial intelligence techniques to aid a user in understanding the data calculated by ACE. See [9] for details.

2.3. ACE Prototype

We have implemented the ACE conceptual framework and its associated services as a web application using the Python-based Django web application framework (`djangoproject.com`). The main components of our software prototype are shown in Fig. 5. A typical three-tier architecture is used with a model-view-controller pattern being implemented by the Django framework. The conceptual framework is implemented as a set of models, with Django handling the details of synchronizing model information to and from the database. (Django can work with a wide range of open-source databases.) Services are implemented via a combination of Django views and client-side Javascript.[1] For instance, when a user edits a workflow, a workflow view is used to generate the initial page that displays a graphical view of the workflow. Client-side Javascript then handles the user's interactions with that page until the user is ready to submit their changes back to the server. A Django controller receives those changes, validates them, and then updates the appropriate workflow model, which in turn gets saved to the database.

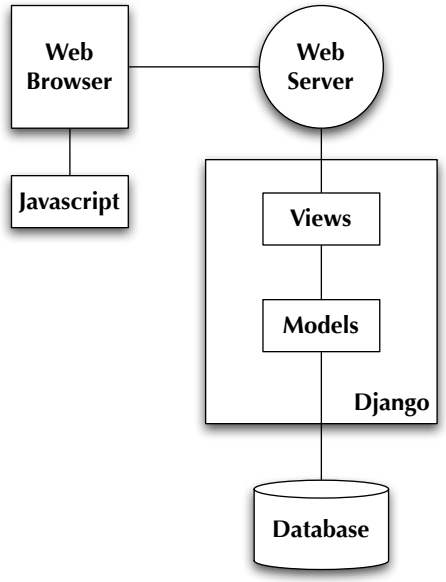

Figure 5. ACE is implemented as a web application using the Django web application framework.

3. ACE Software Architecture

Software architecture is a subfield of software engineering research that looks at the overall structure of a software system and how particular arrangements of a system's components can achieve various non-functional properties [7]. We now discuss how the ACE software architecture supports flexibility, security, safety, and scalability.

3.1. Flexibility

The need for flexibility was apparent early in our interactions with our team of geoscientists. Their problem domain (cosmogenic dating methods) is complex and we had a lot to learn. As a result, the geoscientists taught us about their domain in small pieces, slowly revealing more and more detail about the domain and what they needed from a design environment for cosmogenic dating. As we were not receiving complete information, we needed to make decisions that would allow us to create a prototype quickly while also allowing us to respond to change requests without having to completely discard the work invested in previous iterations. Furthermore, we needed to produce a system that our geoscientists can continue to use once this project ends.

One approach that provided us with flexibility early on was to keep as much of the conceptual model in generic data structures (lists, hash tables, etc.) as opposed to developing specific classes such as `Sample`, `Landform`, `Workflow`, etc. This decision allowed us to quickly alter our data structures as we gained understanding of the problem domain.

A more important decision was to create a mechanism that allows our users to specify the characteristics of a required data set and then allows them to create instances of that data set at runtime. Logically, our solution is similar to research we have performed in the domain of structural computing [1]. In that work, this functionality is achieved by creating three classes: `DataTemplate`, `DataItem`, and `DataCollection` (Fig. 6). The `DataTemplate` class allows users to specify the structure of a data set. For instance, a data set that tracks the position of earth's geomagnetic pole in time will contain three fields: an age, a latitude, and a longitude. The ACE data collection editor allows a user to create a "pole position" template specifying these three attributes. Once this template is created, a user can import a comma-separated file containing data in this format and the template is used to create an instance of a collection that contains `DataItems` featuring the specified fields.

When implementing this idea within Django, we reduced the three classes to two Django models that correspond to tables in a database and provide the same functional-

[1] A Django view is a controller bound to a particular URL within a web application. A view is responsible for generating a response when a browser accesses its URL; typically the Django template system is invoked by a view to generate an HTML page that is sent back to the browser for display.

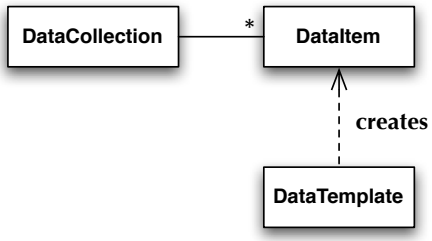

Figure 6. ACE provides the DataTemplate class to allow its users to specify the structure of a data set. A template can create DataItem instances (stored in a DataCollection) to hold the values of a data set.

ity. `TemplateField` is used to define the structure of a data template. Each row in its table represents a single field in a template and specifies the name of the template to which it belongs, the name of the field, and the field type (string or float). Data about a template is constructed by selecting all the rows from the `TemplateField` table that share a template name. Collections are defined with the `CollectionData` model. Each row in this table represents a single attribute on a single piece of data in a collection. A row in the `CollectionData` table specifies a collection's name, the field in the template to which it belongs (via a foreign key relationship), the value of the attribute as either a string or a float, and an index to indicate which piece of data this field belongs to. Collections are reconstructed by selecting all the pieces of data that have the same collection name; individual pieces of data are then reconstructed by looking for entries that share the same index.

Because of this technique, the users of ACE can easily specify and import any data set that might be required for future cosmogenic dating algorithms. The current algorithms within our environment make use of seven different data sets including sea level data, geomagnetic pole positions, magnetic field intensities, etc.

The third decision we made to support flexibility in the ACE design environment was with respect to the design of the calibration and dating workflows. In particular, we needed to provide our users with the ability to create a "workflow skeleton" in which some components of the workflow are left unspecified. Instead, specific components are hooked into "component placeholders" that allow a user to customize a workflow for a particular experiment. Each placeholder is related to a scaling factor and each scaling factor can take on different values. For each value of a scaling factor, a different component can be plugged into the workflow at the point of the placeholder influencing how that particular scaling factor is calculated for that experiment. As an example, one type of scaling factor for cosmogenic dating algorithms is based on sea level data. There are two sea level data sets used by our collaborators, one based on data collected by Fairbanks [4] and one based on data collected by Shackleton [10]. In our environment, we have three components created for calculating the effects of sea level data on a cosmogenic dating algorithm. One component does nothing and is inserted into a workflow if the user selects "Ignore Sea Level Data" as the value for the Sea Level scaling factor in an experiment. The other two components perform the same calculations but differ on which sea level data set is used. If a user selects "Fairbanks" for the Sea Level scaling factor then the component that uses the Fairbanks data set is inserted into the workflow; the Shackleton component is selected in a similar fashion. The placeholder concept becomes quite powerful when you consider that cosmogenic dating algorithms can have multiple scaling factors including taking into account the effects of sunspot data, the morphology of a particular landform, the earth's magnetic field strength, etc. Our approach allows a geoscientist to create a workflow that contains placeholders for each scaling factor that they want to study and they can then create one experiment for each combination of scaling factors that are of interest.

A fourth decision to provide flexibility in the ACE design environment was to establish a generic design for our plotting services that allow new types of plots to be added by simply developing a plug-in that defines the plot's inputs and produces output in a standardized format. The plotting framework makes use of libraries that emulate behavior found in the popular application MATLAB (`http://www.mathworks.com/products/matlab/`) that allows our users to reuse their experience with that application in quickly producing new types of plots for the ACE design environment. Indeed, our collaborators have already developed code that produces a plot that is useful for understanding the erosion of a certain types of landforms and we are currently converting that code into a plug-in for our plotting framework.

The final decision that we made to provide flexibility in the ACE design environment was to allow our team of geoscientists to create new components for the design environment using the Python scripting language. Our first implementation of the design environment used the Java programming language and the use of Java forced our team of geoscientists to write code in a style that was different from what they were used to, which was typing formulas into Excel spreadsheets. By switching to Python, we created a skeleton component that they can customize to produce a new component. In particular, this skeleton has a method called `processSamples()` that provides them access to the current set of samples being processed by a workflow. They can then write code in a syntax that is more

intuitive than what they were used to in Excel and they no longer have to deal with the quirks of Excel programming. Our collaborators are quite comfortable with this set-up and have been producing new components to be included in the design environment on a regular basis.

3.2. Security and Safety

We have made several decisions at the architectural level that impact the security and safety that the design environment provides its users. The first decision relates to the security of our users' data. By deploying the software prototype in a lightweight configuration, we allow our users to run the entire system on a single machine. As a result, even though the design environment is built as a web application, it can be run without an Internet connection and all of the data that a user stores within ACE remains on that machine under their control. This decision is in direct contrast to the approach taken by the CRONUS-Earth series of calculators, as discussed in Section 1. These calculators are run as traditional web applications, on servers outside the control of the software's users; users may be unwilling to submit unpublished data to this application since they do not own the server and cannot look at the application's code to determine if their data will be retained on a third-party server.

The second decision impacts the safety of a user's data by the addition of a number of services that deal with the export of information from the ACE database to a number of widely-supported data formats (such as comma-separated data files). The ability to export data from the database makes it easier to replicate that information across machines and facilities versioning of that information by configuration management systems. Of course, care must be taken to keep this exported data secure, but if a group regularly exports its data and stores that data in a version control system, they will be protected against database failures.

Finally, the design environment supports three types of users with different access rights. A basic user can load new samples into the design environment and run experiments. An advanced user can also create new experiments and workflows. Finally, an admin user can perform the same operations as basic and advanced users but can also add new components, define new scaling factors, add users to the system, and the like. These precautions ensure that only users with deep experience of the problem domain are able to alter the nature of the design environment via the construction of new components, scaling factors, workflows, etc.

3.3. Scalability

The ACE software architecture supports scalability in a number of ways. The first is that the logical architecture shown in Fig. 5 can be mapped into a number of physical configurations. Our users in particular are interested in using this architecture in a configuration that runs all of the ACE components on the same machine. In our implementation, we selected a number of lightweight components—Firefox, lighttpd, Django, sqlite—to play the roles of browser, web server, web application framework, and database respectively. Each of these components have relatively small disk and memory footprints that make it feasible to run everything on a single machine with reasonable performance.

However, the architecture is flexible enough that we can switch from this single-machine configuration to an extremely distributed configuration in which the components for each of these roles are run on different machines, or any combination in between. In the distributed setting, ACE switches from a single user tool to one that can support small research teams collaborating on a local area network or to one that supports multiple research teams all sharing published samples and workflows with each other. In addition, if a research team decides to scale the implementation to handle larger groups of users, they can transparently switch from the lightweight components used in the initial implementation to more heavyweight components, in particular switching from lighttpd to Apache and switching from sqlite to databases used in production settings.

Of course, switching from a single machine configuration to one that spans multiple machines can impact other architectural characteristics. While a multiple machine configuration provides scalability, it becomes more difficult to maintain security. In particular, the security techniques that need to be applied will change as the number of machines and the number of users increase. In this configuration, it will be more difficult for a user to store all of their data on their own machine; instead, the data for an entire research group will be shared on a single server and the group's system administrator will need to have experience in correctly configuring the shared database to provide secure access to authorized users. Fortunately, there are a number of best practices that can be adopted when a research group needs to run ACE in a distributed setting. The important point here is that ACE is designed in such a way that it can support both single user/machine and multiple user/machine configurations and each research group can pick and choose how they wish to deploy ACE to best fit their needs.

4. Evaluation

At the beginning of this project, our collaborators provided us with an Excel spreadsheet that implemented a single cosmogenic dating technique for samples containing the nuclide ^{36}Cl. The spreadsheet could only process one sample at a time and had its production rates hardwired into it.

That is, the spreadsheet did not contain any functionality to handle the process of calibration.[2] Furthermore, the dating algorithm implemented by the spreadsheet had a certain set of scaling factors built into it with no flexibility to change their algorithms or to add or remove new scaling factors.

Our design environment can now be used to design an experiment that exactly mimics the functionality of the original spreadsheet. That configuration, however, is just one of an almost unlimited set of experiments that can be specified, executed, and analyzed by the current design environment. The flexibility built into our concept of workflow allows any number of calibration and/or dating algorithms to be constructed, each of which can support multiple combinations of scaling factors and other parameters and variables. A particular scaling factor can easily be adjusted by adding a new "mode" to it and supplying a component that performs the calculations associated with that mode. Our design environment smoothly handles the process of calibration and ensures that dating workflows use production rates calculated by calibration workflows that have the same structure as the dating workflows and the same set of scaling factors. In addition, the design environment can store multiple sets of samples and can apply a dating workflow to multiple samples at once. Furthermore, our design environment provides plotting and analysis services over the samples that it has processed, allowing our users to perform productive work within the environment. Finally, it allows users to export its data for postprocessing or dissemination.

The "look and feel" of the ACE design environment has been greatly influenced by the feedback of our team of geoscientists. As ACE is a web application, we hosted a version of the ACE prototype on a private server and allowed the geoscientists to use this software during development. As new features were finished, or the user interface was enhanced or changed, new versions were deployed to the private server so the geoscientists could try out the latest version of the design environment. They would then provide feedback which we would attempt to incorporate in subsequent releases. This incremental, iterative process is exactly the type of development life cycle that is recommended when attempting to create a design environment for a particular community of users [5].

As a result of these design, implementation, and evaluation choices, our cosmogenic dating environment demonstrates how modern software engineering principles and techniques can be used to produce flexible and extensible software that meets the needs of its users. Our initial evaluation has compared the capabilities of the ACE design environment with the functionality of the software it was built to replace (as well as with other cosmogenic dating software). It represents a proof-of-concept that the limitations of these previous systems were artificial and that modern software engineering techniques can be used to produce a system that provides the flexibility needed by geoscientists to perform cosmogenic dating research. No other cosmogenic dating software offers the same functionality, flexibility and future extensibility as is provided by ACE.

In the future, we intend to engage in more formal evaluations of our work on ACE, including usability studies of ACE's user interface as well as studies of whether geoscientists can make full use of ACE's extensibility options.

[2] Indeed, this caused confusion after our first attempt to build a Java-based design environment for our collaborators. During that development effort, we had thought of the production rates as constants that had been plucked from a standard textbook. We were astonished to find out that, no, those rates had been calculated by a calibration algorithm and that we now had to absorb a whole host of previously unknown requirements related to the process of calibration!

5. Related Work

Our work on ACE is most closely associated with the research performed on design environments in general. Work on design environments evolved from an interest on how to improve the functionality provided by programming environments [12]. Design environments are characterized not only by functionality to perform a particular task but also by the services they provide for a user to reflect on the task itself. ACE not only calculates ages based on measured inventories of cosmogenic nuclides, it also provides functionality to analyze and compare the results of cosmogenic dating techniques allowing researchers in this domain to refine existing techniques and design new ones. Design environments have also been used to help users understand and apply standardized techniques within a domain. For instance, Garlan et. al [6] developed techniques for customizing software architecture design environments with information about particular software architectural styles. These customizations allowed a user to more quickly create software systems that followed these styles. In ACE, we have carefully designed the ways in which users can extend the design environment, allowing them to add new data sets, new components, new scaling factors, and new plot types as their needs evolve. These standardized extension points will allow experts in this problem domain to teach novices (e.g. graduate students) the ways in which cosmogenic dating techniques can be altered and evolved. None of the other software systems for cosmogenic dating (see Section 1) offer the flexibility and extensibility provided by ACE.

Our work on defining the ACE conceptual framework as well as the ACE software architecture is similar to work performed on domain-specific software architectures [11]. The conceptual framework identifies the key concepts required to adequately model the information, functionality, and services encountered in the domain of cosmogenic dating techniques. The decisions we made at the architectural level of our design environment ensure that important non-

functional properties (flexibility, scalability, and security) are enabled that will provide users in this community with the confidence they need to adopt the software and incorporate it into their future research efforts.

Finally, ACE easily matches the functionality provided by the cosmogenic software discussed in Section 1. For instance, the CRONUS-Earth series of calculators performs calculations with respect to the ^{10}Be and ^{26}Al nuclides. ACE handles those nuclides plus ^{14}C, ^{21}Ne, ^{3}He, and ^{36}Cl and via the nuclide editor can be configured to handle others. Indeed, it is ACE's support for extensibility that allows it to offer features and services that are not found in any other cosmogenic dating software package. Finally, while the primary focus of this paper has been on the software engineering aspects of our work, we should note that the components and workflows included in ACE incorporate a wide array of current research in cosmogenic dating [2,3,8].

6. Conclusion

The ACE design environment represents a significant advance in the capabilities provided by cosmogenic dating software. It provides geoscientists with the ability to both evaluate existing cosmogenic dating techniques and design new ones. It is based on a comprehensive conceptual framework that ensures that all aspects of cosmogenic dating techniques are modeled by the system. Its software architecture provides flexibility to allow researchers to explore "what if" scenarios on how these techniques can be evolved, provides scalability to support both individual and group work, and provides security to keep its users' data safe and under their control. Furthermore, we are using AI techniques to augment the environment with analysis capabilities that are useful to expert researchers while also providing novices with insight into the kinds of techniques and services that are required to do research in this area.

Our future work on ACE will evolve its capabilities in a number of ways. We will expand its support to perform experiments that involve more than just one nuclide for a particular sample. In addition, we will work with our collaborators to understand the kinds of services they need to take the information they have calculated about individual samples and use it to understand the evolution of the samples' associated landform. For instance, we may allow users to add components that model the erosion patterns of a particular type of landform and then match the predictions of this model with the data gleaned from samples taken from such a landform. This shift in focus from individual samples to entire landforms will increase the value of ACE for this research community and enable the next generation of research in the field of cosmogenic dating.

7. Acknowledgment

This material is based upon work sponsored by the NSF under Grant Number ATM-0325812 and Grant Number ATM-0325929.

References

[1] K. M. Anderson. Towards lightweight structural computing techniques with the smallsc framework. In *Proceedings of the 2005 Symposium on Metainformatics. ACM International Conference Proceeding Series*, volume 214, pages 1–10, 2007.

[2] D. Desilets and M. Zreda. Spatial and temporal distribution of secondary cosmic-ray nucleon intensities and applications to in-situ cosmogenic dating. *Earth and Planetary Science Letters*, 206(1):21–42, 2003.

[3] D. Desilets and M. Zreda. Elevation dependence of cosmogenic 36cl production in hawaiian lava flows. *Earth and Planetary Science Letters*, 246:277–287, 2006.

[4] R. Fairbanks. A 17,000-year glacio-eustatic sea level record: influence of glacial melting rates on the Younger Dryas event and deep-ocean circulation. *Nature*, 342:637–642, 1989.

[5] G. Fischer, R. McCall, J. Ostwald, B. Reeves, and F. Shipman. Seeding, evolutionary growth and reseeding: Supporting the incremental development of design environments. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pages 292–298, 1994.

[6] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 175–188, 1994.

[7] R. Kazman, P. Clements, and L. Bass. *Software Architecture in Practice*. Addison-Wesley, 2nd edition, 2003.

[8] N. A. Lifton, J. W. Bieber, J. M. Clem, M. L. Duldig, P. Evenson, J. E. Humble, and R. Pyle. Addressing solar modulation and long-term uncertainties in scaling secondary cosmic rays for in situ cosmogenic nuclide applications [rapid communication]. *Earth and Planetary Science Letters*, 239:140–161, Oct. 2005.

[9] L. Rassbach, E. Bradley, K. Anderson, M. Zreda, and C. Zweck. Arguing about radioisotope dating. In *Proceedings of the International Workshop on Qualitative Reasoning about Physical Systems*, 2007.

[10] N. J. Shackleton. The 100,000-year ice-age cycle identified and found to lag temperature, carbon dioxide, and orbital eccentricity. *Science*, 289:1897–1902, 2000.

[11] W. Tracz. DSSA (domain-specific software architecture): Pedagogical example. *ACM SIGSOFT Software Engineering Notes*, 20(3):49–62, 1995.

[12] T. Winograd. From programming environments to environments for designing. *Communications of the ACM*, 38(6):65–74, 1995.