

Annotating High Performance Computing Simulations with Semantic Metadata

Dylan Stark

Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana 70803
Email: dstark@cct.lsu.edu

Gabrielle Allen

Department of Computer Science
Louisiana State University
Baton Rouge, Louisiana 70803
Email: dstark@cct.lsu.edu

Abstract—We describe the design and use of an ontology for describing the Cactus framework — a component based architecture for scientific computing. This work forms a starting point for the use of ontologies to support new component applications involving multiple scale and models, realtime assimilation of sensor data and deployment on distributed Grids.

I. INTRODUCTION

Scientific application development is becoming increasingly complex as we strive to better model real world systems, while taking advantage of developments in computer hardware as well as new paradigms such as dynamic data assimilation from distributed sensors. The trend is towards supporting multiple types of physics, different temporal and spatial scales, numerical methods, I/O libraries, visualization interfaces, and more. Likewise, grid computing and sensor networks are also adding to the complexity of application development. Grid computing involves interconnected compute resources and data archives, with orchestrated application workflows across them. Incorporating data from sensor networks requires that our applications need to include algorithms and technologies for dynamic data driven systems.

Component architectures [6], [16], [3] are one way to assist application developers, but the modularity and interoperability of software components introduces new information management challenges. There is a need to keep track of all of the different available components and how they are used together. This information about the software components also needs to be associated with compute and data resources as well as where the applications are run and the output they produce. Having this information available to the developers and end users will enable applications to be better matched with compute and data resources.

A problem seen with the proliferation of components and subsequent (composed) applications is managing all of the associated information. Each component carries information about its developers, version, etc. as well as higher-level descriptions of what the component does. A composed application carries similar information: the function of a component application is the aggregate of the functions of its components. To support information management in scientific computing, we need a common language for expressing ideas and concepts, ways of identifying the individuals/instances

in the system, and interlinking the data. The common language will let us share the concepts such as *black holes*, *Cactus applications*, and *publications*, between groups and organizations which may otherwise never interact. The ability to identify individuals is necessary so that we can uniquely reference particular *applications* or *resources* when applying metadata. Finally, linking data will allow for data aggregation in a system where no central information source is available, or even practical.

II. BACKGROUND

A. The Cactus Toolkit

Cactus is a robust component framework with an active user base [6]. Components encapsulate, or implement, some functionality. Examples of available components include parallel drivers, using MPI and OpenMP, numerical solvers, and even an HTTP server, for web-enabling an application. In Cactus, these components are called *thorns* and the core, which provides for the composition of thorns, is called the *flesh*. The make system collects thorns, along with compile-time options, to generate a configuration for a target machine architecture. A configuration is then compiled to produce a build.

Tools already exist for managing metadata associated with Cactus applications, including the thorn management package XCactus [18] and the Formaline [5] thorn which publishes metadata from a running simulation. What the current tools lack is a consistent information architecture to support them. Each one defines its own syntax, either as key-value pairs or some dialect of XML. There is no ready solution for exchanging and/or integrating information between tools. Also, each of these approaches lacks the ability to model information about Cactus applications in a way that is extensible.

B. Semantic Web Technologies

The semantic web initiative is aimed at developing a machine-processable environment of linked data in order to promote collaboration and information sharing on the WWW. The two foundational layers of the semantic web stack [12] are a data model, RDF, and an ontology language, OWL [11], [17]. There is also a growing collection of tools and (W3C) standards behind the semantic web, such as the SPARQL query language [14], as well as libraries such as RDFlib [10]. Large

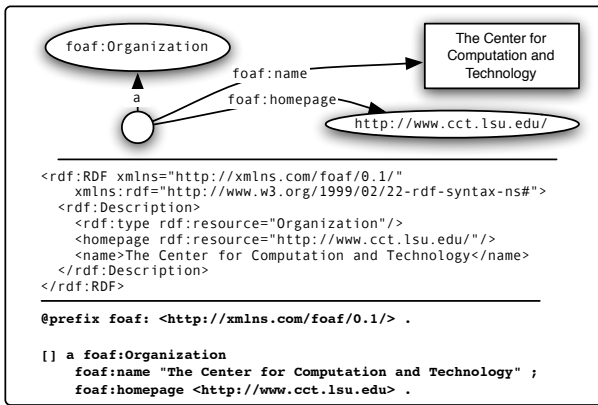


Fig. 1. A comparison of RDF/XML (middle) and Notation3 (bottom) representations of the top RDF graph.

corporations including IBM, HP, and Oracle are expressing interest in the potential of these technologies by providing production quality datastores and frameworks [7], [8], [13].

RDF is a domain-independent model that lets users describe resources using their own vocabularies. No assumptions are made about any particular application domain, nor does it define the semantics of any domain. The user defines semantics in an RDF Schema¹ or an OWL ontology. Ignoring references to the FOAF ontology [2] in Fig. 1, the RDF simply expresses that there is *something* (the empty node) which relates to two other resources (*things*) and the literal string “The Center for Computation and Technology”. The FOAF terms `name` and `homepage`, provide the real semantics; i.e. that there is an organization (the empty node) which is named “The Center for Computation and Technology” and has its homepage located at `http://www.cct.lsu.edu/`.

The ontology language provides a means for defining the terminology necessary for describing a domain of interest. The fundamental elements of OWL are classes and properties. An OWL class is a set of individuals, or instances, of a concept. Properties are used to define relationships between individuals. The terms defined in an ontology can be used as labels in an RDF graph. The previous RDF example (see Fig. 1) defined an individual (the empty node) of the class of organizations and related it to its name and the URI of its homepage. The labels `foaf:Organization`, `foaf:name`, and `foaf:homepage` are all terms defined in the FOAF ontology, which was designed for describing people.

III. CCTK-O: THE CACTUS ONTOLOGY

Component architectures help to deal with the complexity inherent in scientific application development. The approach has introduced new complexities, though, and turned the development process into an information management problem. The developer has given up some control by using code developed by others. Two questions which arise now are:

¹RDFS allows the representation of some ontological knowledge such as: class hierarchies, properties, domain and range restrictions and instances of classes.

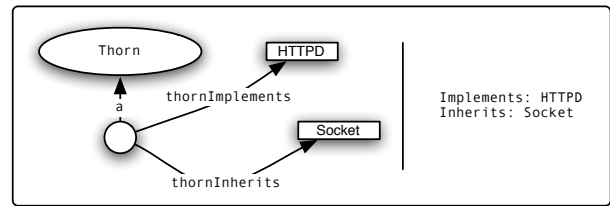


Fig. 2. Thorn metadata extracted from CactusConnect/HTTPD/interface.ccl. This denotes an unnamed thorn which implements HTTPD and inherits from Socket.

what is out there, and what works together? The *CCTK-O* ontology [15] was designed as part of an effort to utilize semantic web technologies for information management in Cactus. The goal is to enable answering those two questions by providing a flexible and extensible infrastructure for describing applications built with Cactus. The *CCTK-O* uses OWL-DL to describe (define terminology for) Cactus in a formal, machine-accessible way, and uses RDF to allow for the encoding of metadata in a decentralized fashion that maps well onto the distributed component model.

The design of the *CCTK-O* ontology consisted of term extraction from existing documentation, source code, and technical papers. Email exchanges with the core Cactus maintainers were helpful in clarifying the intended meaning of certain terms. As an example, Figure 2 show the RDF graph generated for a particular interface. Conceptually, the generated RDF statements convey that there is some thorn which implements HTTPD and inherits from another implementation of Socket. Specifically, the thorn which implements an HTTP daemon inherits properties from a thorn that implements some socket interface. The terms `Thorn`, `thornImplements`, and `thornInherits` are defined in the the ontology and correspond to the concepts in Cactus². In total, the *CCTK-O* ontology includes around 38 classes and 44 properties.

IV. USE CASES

We illustrate the *CCTK-O* with two example projects showing how the OWL specification has been used. Each project satisfies a need in component architectures and scientific computing, uses the RDF data model for metadata, and relies on the *CCTK-O* for providing a structured vocabulary for the metadata.

A. Component Registry (*The Thornery*)

A light-weight thorn repository was developed to demonstrate automated extraction of RDF from Cactus source code and to show how that metadata could support information management efforts.

Component development is a largely decentralized effort. Cactus coordinates and interfaces individual thorns, but many separate groups develop thorns. A central *registry* for collecting information about different thorns will aid users looking for available modules. This registry will be maintained by

²Full coverage of the such concepts can be found in the Users’ Guide [1].

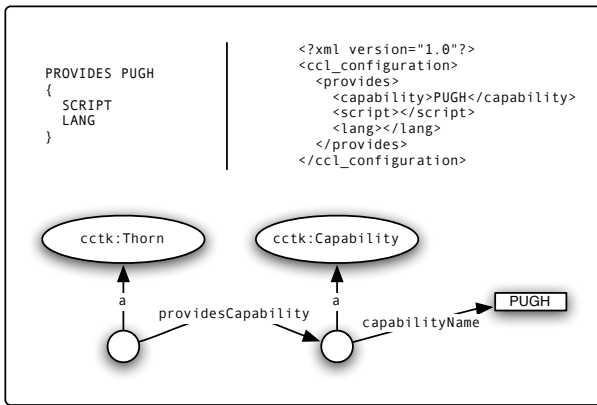


Fig. 3. A cross-section of CCL, XML, and RDF. The XML was generated by an XCactus CCL parser, and the RDF was generated from the XML to speed up production.

the core Cactus team to provide an authoritative place to find metadata relating to Cactus thorns. Note, however, that there is no intention for this to be a definitive collection of metadata; in fact, the use of semantic web technologies for modeling information is meant to encourage other, interoperable, efforts.

The Thornery was implemented as a RESTful web service [4] which serves metadata about available thorns. It also provides a URI infrastructure for the core thorns in the Cactus toolkit. Every thorn must define the appropriate Cactus Configuration Language (CCL) files. These files define those characteristics of the thorn necessary for the framework to correctly use it. As such, the metadata provided by the CCL files is used to describe the thorn. Fig. 3 shows a cross-section of the CCL for a `configuration.ccl` file, alongside the XML and RDF produced from it. The database for the registry consists of the collection of such RDF data.

As for the interface provided by the Thornery, each loaded thorn is given a dereferenceable URI. Consequently, all available information for a thorn can be returned by accessing that URI. Also, one aspect of RESTful web services is content-negotiation, where a single URI can be used to request many different representations (HTML, XML, RDF). The Thornery stores all metadata in RDF, described using the *CCTK-O*, as a base representation. Converters are available for transforming this metadata into the appropriate representations in HTML, XML, and JSON.

B. Automated Development (OGRe)

Given a problem specification, ie. some desired functionality, and a set of components, it is far from trivial to arrange the components to compose an application with some desired characteristics. First, the set of components may be incomplete, with missing support modules. Second, the developer may not know how to set initial values for some parameters provided by the components. A new user developing an application would benefit from a tool that, when given a set of thorns, suggests what additional thorns might be missing. This would require a system with a knowledge base of available

thorns and known application configurations. Also, the system would need to be able to resolve inter-thorn dependencies, and to determine valid configurations which are a *best* match. The *OGRe* case-based reasoner [9] discussed next was developed to support this scenario.

OGRe is a light-weight case-based reasoner for building Cactus applications. Case-based reasoning for developing component-based applications is a natural fit since new applications will reuse many of the components used in previous applications and it will allow the developer to build on past experience.

Users are able to write problem specifications, descriptions, as parameter files that list the thorns which the developer knows are required. The RDF representation of the parameter file is then generated and used in the matching process. In this approach, the developer only needs to partially specify a Cactus application (as a parameter file), and the *OGRe* will attempt to complete it for them. Parameter files should already be familiar to Cactus users, which could help lower the barrier to entry for using this system. The *OGRe* works completely off of the metadata annotations of the components and application files, meaning there are no special formats necessary for utilizing this case-based reasoner.

This approach is light-weight because it reuses/couples available semantic web tools to orchestrate a system which functions as a case-based reasoner. Cactus metadata, described in RDF using the *CCTK-O*, is used to populate a knowledge base of known applications. There is already a large collection of production code available for use in the knowledge base.

V. CONCLUSIONS & FUTURE WORK

We have described the *CCTK-O* ontology as a necessary set of terms for modeling metadata for the Cactus Framework. Our initial focus has been on expressing metadata that can be extracted automatically from source code, with no intervention from users or developers. We also described initial tools built on top of the resulting ontology: a component registry and a case-based reasoner for application development.

Our metadata description is now being extended to describe dynamic runtime data from Cactus, paving the way for dynamic data driven application scenarios able to exploit changing computational Grids and the assimilation of sensor and experimental data. Although our motivating applications are those using the Cactus Framework, these tools and design methodology should be applicable to many component-based frameworks, such as the Common Component Architecture (CCA).

ACKNOWLEDGEMENTS

We gratefully thank our colleagues in the Cactus Team, particularly Erik Schnetter, Thomas Radke, Robert Engel and Tom Goodale for discussions and advice. Funding from the NSF DynaCode project (#0540374) is acknowledged.

REFERENCES

- [1] G. Allen, T. Goodale, G. Lanfermann, T. Radke, D. Rideout, and J. Thornburg, *Cactus Users' Guide*, 2004, <http://www.cactuscode.org/Guides/Stable/UsersGuide/UsersGuideStable.pdf>.
- [2] D. Brinkley and L. Miller, "FOAF vocabulary specification 0.9: Namespace document 24 may 2007 - 'rehydrated' edition," May 2007, <http://xmlns.com/foaf/spec/20070524.html>.
- [3] "Earth system modeling framework," <http://www.esmf.ucar.edu/>.
- [4] R. Fielding and R. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.
- [5] "The formaline thorn," <http://cactus.cct.lsu.edu:5555/Thorns/Formaline/>.
- [6] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, "The cactus framework and toolkit: Design and applications," in *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*. Springer, 2003, <http://www.springerlink.com/content/2fapcbeyyclxg0mm/>.
- [7] "Ibm boca users guide," <http://ibm-slrp.sourceforge.net/wiki/index.php/BocaUsersGuide-2.x>.
- [8] "Jena semantic web framework," <http://jena.sourceforge.net/>.
- [9] J. Kolodner, *Case-based reasoning*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1993.
- [10] "Rdf:lib: Home," <http://rdf:lib.net/>.
- [11] "Resource description framework (RDF) / W3C semantic web activity," <http://www.w3.org/RDF/>.
- [12] "Semantic web layer cake," http://www.w3c.rl.ac.uk/pasttalks/slidemaker/KM_Europe/slide5-2.html.
- [13] "Sesame rdfstore," <http://www.openrdf.org/>.
- [14] "Sparql query language for rdf," <http://www.w3.org/TR/rdf-sparql-query/>.
- [15] D. Stark, "Cactus computational toolkit (CCTK) ontology: Namespace document 26 march 2006 - (first edition)," Mar. 2006, <http://www.cct.lsu.edu/dstark/cctk/0.1/>.
- [16] "The common component architecture forum," <http://www.cca-forum.org/>.
- [17] "Web-ontology (WebOnt) working group (OWL) (closed)," <http://www.w3.org/2001/sw/WebOnt/>.
- [18] "Xcactus homepage," <http://wugrav.wustl.edu/people/JTAO/research.php?page=2>.