

DISTRIBUTED COMPUTING FRAMEWORK FOR SYNTHETIC APERTURE RADAR APPLICATION

Eric M. Gurrola, Paul A. Rosen
Radar Science and Engineering Section
NASA Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA, USA

Eric.M.Gurrola@jpl.nasa.gov, Paul.A.Rosen@jpl.nasa.gov

Michael Aivazis
Center for Advanced Computing Research
California Institute of Technology
Pasadena, CA, USA
Aivazis@caltech.edu

Abstract— We are developing an extensible software framework, in response to Air Force and NASA needs for distributed computing facilities for a variety of radar applications. The objective of this work is to develop a Python-based software framework, that is the framework elements of the middleware that allows developers to control processing flow on a grid in a distributed computing environment. Framework architectures to date allow developers to connect processing functions together as interchangeable objects, thereby allowing a data flow graph to be devised for a specific problem to be solved. The Pyre framework, developed at the California Institute of Technology (Caltech), and now being used as the basis for next-generation radar processing at JPL, is a Python-based software framework. We have extended the Pyre framework to include new facilities to deploy processing components as services, including components that monitor and assess the state of the distributed network for eventual real-time control of grid resources.

I. INTRODUCTION

The Air Force Research Laboratory (AFRL) has entered into a collaboration with NASA/JPL in the area of emerging computational technologies, including the area of grid computing. Building on successes in other projects related to on-board radar processing technology between AFRL and the Jet Propulsion Laboratory (JPL), this collaboration seeks to extend the radar processing methodology from specific computational elements implemented on a special purpose processor such as a Field Programmable Gate Array (FPGA) or other embedded system to a distributed computing environment. This computing environment could extend far beyond traditional computing centers to small, embedded computing systems that deliver new capabilities. NASA space exploration missions, such as the Mission to Mars, provide scenarios of application. Within this domain, emerging areas of computing will be challenged to deliver highly embedded solutions meeting size, weight, and power constraints yet able to integrate with flexibility and robustness into a computational grid extending beyond global proportions.

Our objective is to develop a Python-based software framework, that is, the framework elements acting as middleware to allow developers to control processing flow on a grid in a distributed computing environment. Framework architectures to date allow developers to connect processing functions together as interchangeable objects, thereby allowing a data flow graph to be devised for a specific problem to be solved. The Pyre framework, developed at the California Institute of Technology (Caltech), and now being used as the basis for next-generation radar processing at JPL, is a Python-based software framework. The goals of Pyre framework development are extremely well-aligned with the envisioned distributed computing applications to synthetic aperture radar.

II. SCENARIOS FOR APPLICATION TO INTERFEROMETRIC SYNTHETIC APERTURE RADAR

Characterization of the way the surface of the earth deforms in response to internal and external forces relies on a variety of geodetic measurement techniques. Most of these methods provide point-to-point measurements of the displacement field. In contrast, InSAR provides high-resolution map views of the deformation [1-3]. The spatial resolution provided by InSAR has the potential to revolutionize studies of seismic and volcanic hazards, the relationship between fluid flow in the crust and surface deformation, the dynamics of long term strain accumulation at tectonic plate boundaries, and the relationship between glacier dynamics and climate change.

InSAR has proven itself as a reliable technology for topographic and surface deformation mapping. Aircraft with single-pass InSAR systems routinely generate fine-resolution topographic maps. In these systems, two antennas are separated spatially to form a baseline in a planar surface normal to the flight direction. From the coherent radar images of an area one can construct an interferogram as the product of one image and the complex conjugate of the

other. The phase of each point in the interferogram encodes the difference in the length of the paths traversed by the radar signal in propagating from the radar antenna to the surface and back again. The phase is related to the topography of the surface.

Repeat-pass InSAR relies on repeated imaging of a given geographic location by air or space-borne radar platforms. If the surface is unchanged between imaging times, then the repeat-pass interferometer behaves as a topographic mapper much as the single-pass system. However, if the deformation is such that the surface itself remains intact, such as with tectonic motion, then the phase encodes the topography and displacements of the surface. Sensitivity of the phase to displacements is much higher than to the topography itself. If the surface is disrupted between observations, such as by lava flows, building collapse, or severe shaking, the phase becomes random as the surface becomes decorrelated between imaging times. An interferogram used for measuring surface displacements is a map view of surface motion, with a typical pixel size of 20 meters and a detection threshold of a few millimeters.

Geophysics applications to date have concentrated on map views of surface motion over limited areas (100 km x 100 km typically). Since an interferogram encodes both topography and deformation, data reduction methods must reliably remove the topographic effects. This can be done either using an existing topographic data set, or by generating heights from an interferogram known to include no deformation. Given the appropriate radar data sets and/or

digital elevation model (DEM), and knowledge of the sensor and orbit, generating deformation maps over very wide areas is in principle straightforward.

While InSAR is a method with potential for tremendous growth, exploitation of this tool currently involves individual and isolated data archives, inhomogeneous sets of processing codes, and products that do not merge well with other related data. This is not unexpected with a developing technique, but without further synthesis, organization, and development of standards, the full potential of InSAR to solve both scientifically and socially important problems will not be realized. The field will develop several data sources over the next decade as individual countries and agencies launch their own satellites or embark on other acquisition programs. These data along with intermediate derived products (such as interferograms) will need to be archived and cataloged. It is likely that the producers of the data will set up their own archive facilities, each of which will house many terabytes of data.

We seek to establish a system that will allow researchers to utilize data from any participating archive site in a transparent manner. A typical processing sequence initially consists of a set of basic steps that are repeated each time new parameter choices are made. Some of these steps are simple and well suited to the interactive environment of a workstation, but others are computationally intensive and lend themselves to parallel processing on large machines. At

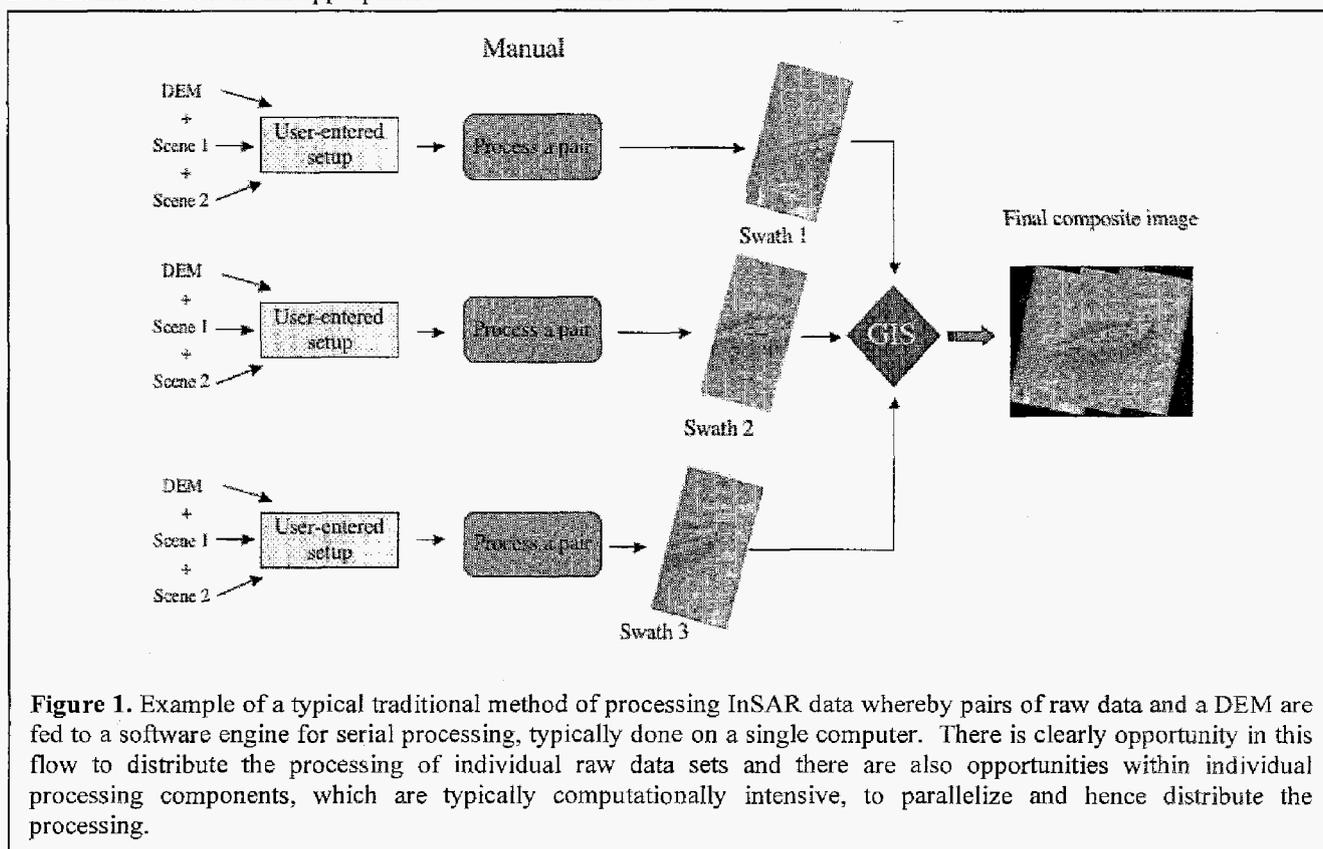


Figure 1. Example of a typical traditional method of processing InSAR data whereby pairs of raw data and a DEM are fed to a software engine for serial processing, typically done on a single computer. There is clearly opportunity in this flow to distribute the processing of individual raw data sets and there are also opportunities within individual processing components, which are typically computationally intensive, to parallelize and hence distribute the processing.

the moment, the processing consists of manually moving the data to the appropriate environment, applying the necessary computation, and then retrieving the data. While this can be somewhat automated through the use of processing scripts, each project tends to involve a custom script. This is a serious impediment to the use of the data by non-experts, and also for the "discovery phase" of data mining, where one may need to scan and/or combine many interferograms looking for subtle changes that are not obvious on any single scene. Figure 1 illustrates a typical traditional processing approach, where sets of data are processed serially to deformation products in particular geographic areas, then merged at a later time through a GIS system. Figure 2 illustrates the desired science-driven approach, in which the details of how the computation is done and which computing resources are used for any specific part of the computation is subsumed into the computational framework

Beyond basic processing tools, there are sophisticated algorithms that are developed for specific scientific applications. However, the lack of an underlying framework for the data manipulation causes each of them to be "one of" pieces of code that are not shared with other applications. The distributed framework will allow the user to specify, in a simple graphical way the processing sequence to be applied to the data, choosing the steps from a suite of available code and plugging them together. The underlying framework will take care of moving the data between various computers and displaying the results on the home workstation.

III. THE COMPUTATIONAL FRAMEWORK

The computational framework is based on Pyre [4], which was developed at Caltech. Pyre is based on Python and provides a modern object oriented layer on top of

legacy software. The Pyre layer allows us to develop abstract components referred to as facilities that specify a user interface and methods. Specific implementations of the abstract facilities can be bound to an application based on data provided by the user. The legacy software, often written as monolithic special purpose code in Fortran or C, is bound to a Python component through standard methods as illustrated in Figure 3. Pyre provides access to many convenience classes to any component written in Pyre, which brings a whole host of services to the legacy software components. The legacy radar processing software named ROIPAC developed at JPL has been re-factored into the Pyre framework.

In the work described here we have re-factored the interface definition for pyre components in such a way that appropriately implemented components can be used as network services. The current implementation requires the developer to write a small wrapper class that derives from an existing, user-supplied component and the framework Service class. The purpose of the wrapper class is to forward network requests to the implementation methods of the user supplied components.

A number of classes were developed for this purpose to enable reliable and safe deployment of components as distributed services. The required classes were the following: (1) authenticating users of services; (2) communication protocols; (3) encoding and transport of binary data across the network; (4) monitoring the status of remote processes; (5) evaluating the status of the network itself for efficient deployment of resources; (6) and providing for persistence and termination of services. Figure 4

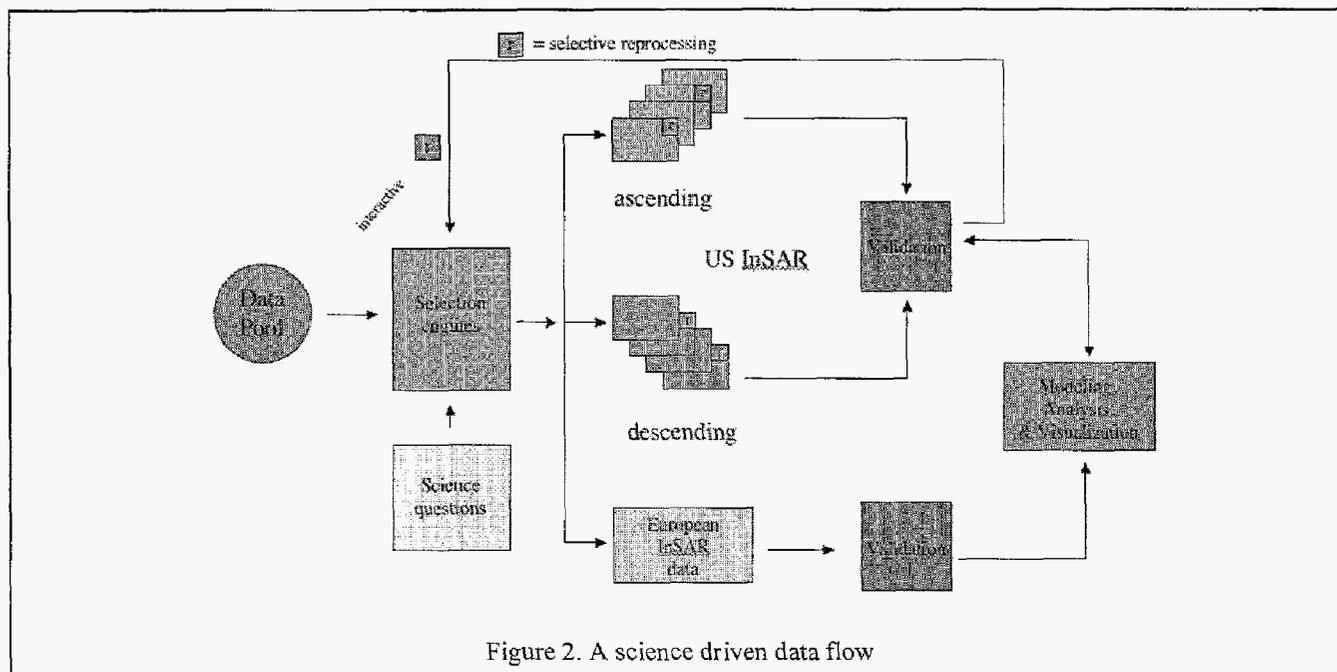


Figure 2. A science driven data flow

- The integration framework is a set of co-operating abstract services

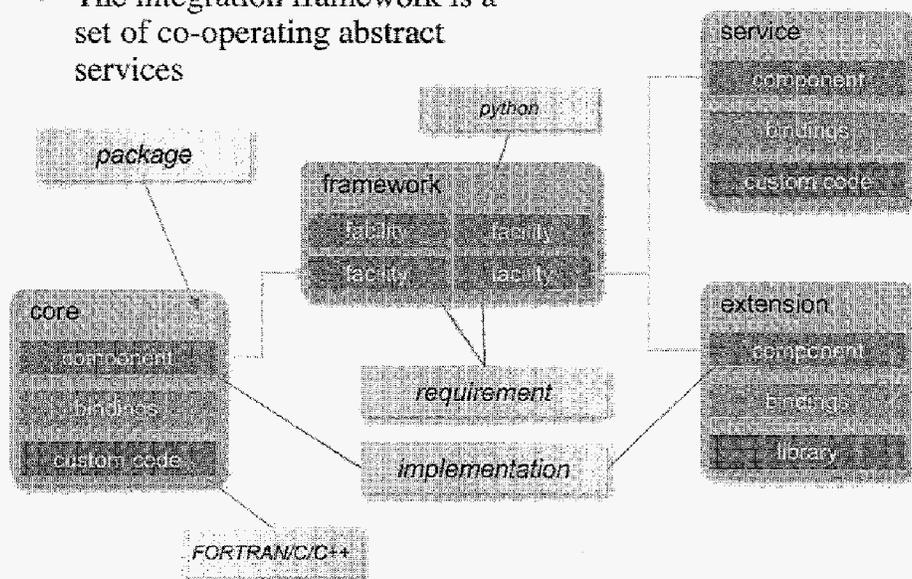


Figure 3. The Python-based Pyre Framework for integrating legacy scientific software into an object oriented framework of interacting services.

indicates the user's perspective of the resulting distributed computing framework.

We have constructed a component named *IPA* that acts as an authentication service for distributed computations. *IPA* listens to the network at a user specified port for incoming authentication connections that provide a user name, password pair that is verified against an internally managed database of known users. The user database can be stored as a flat ASCII file with the passwords encrypted using a variety of encryption algorithms, or it can be stored as a table in a relational database back end such as PostgreSQL. Attempts to connect that do not obey the connection protocol or contain invalid user name-password pairs are ignored. Upon receipt of a valid user name-password pair, *ipa* issues session keys that the client can present to other services as authentication credentials. A variety of algorithms for session key expiration are provided, such as keys that can be used only once, keys that have a finite lifetime and keys that never expire.

The prototype of a persistent service is a component called *IDD*. *IDD* issues tokens in a fixed numeric sequence with the capability of encoding additional information in the token, such as the date it was issued. Component *idd* persists the currently issued token so that upon restart it can continue issuing tickets from the same sequence.

Support for Remote Procedure Calls is implemented by the various subcomponents of the class *Service*. The IP protocol used is determined by whether a component inherits

from *TCPService* or *UDPService*. Support for staging and launching under either protocol is provided, including synchronous and asynchronous operation, consistent with the widest possible applicability of these objects to the construction of real applications. The interprocess communication protocols are implemented as descendants of the base class *Marshaller* that is responsible for specifying the actual protocol in use. The class *Pickler* implements a protocol that employs the built in python capability of object serialization to transport commands and responses between the client and the service. *XMLRPC*, currently under construction, was meant to be proof that multiple protocols can be supported, but recent advances in the interprocess communication strategies employed by the Grid community all focus on SOAP derivatives, making *XMLRPC* potentially obsolete. As a result, this direction has been deemphasized and will only be pursued if there is sufficient end-user demand for it.

The components necessary to encode and transport binary data are under design. Prototypes have been constructed to transfer data between client and server using all UNIX IPC mechanism and IP protocols, and in fact have formed the basis for the operation of the services mentioned in the previous two sections. However, high performance data transport, the kind that would be suitable for large datasets, is still the subject of active research. Access to the data streams from lower level languages is currently provided by populating arrays of native data types and making them available through an API. Support for more advanced data types will be considered on demand as

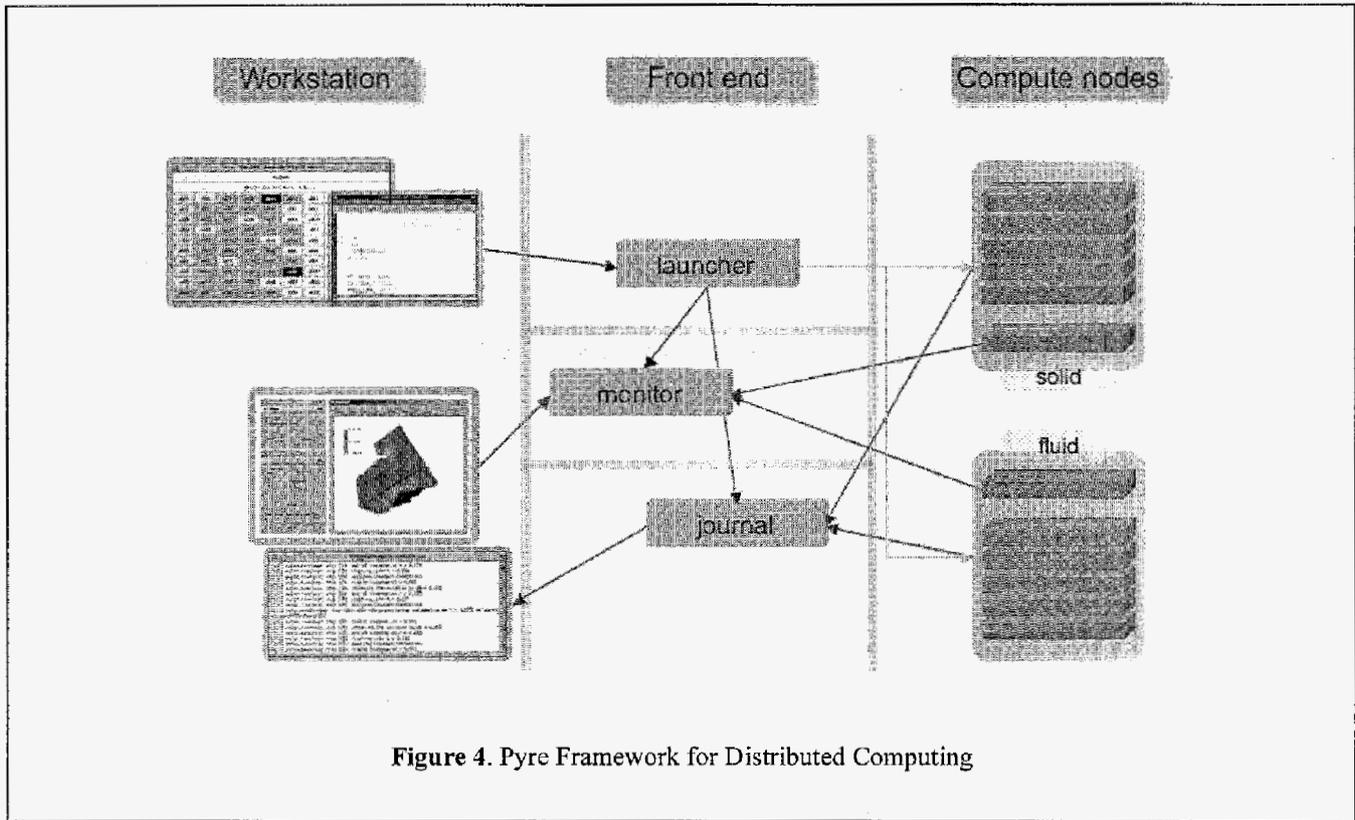


Figure 4. Pyre Framework for Distributed Computing

applications that require such support are constructed.

In order to collect process status from a distributed calculation, we have extended the standard Pyre package journal with the ability to transmit status messages to a monitoring service that is responsible for the collection and collation of messages from all the services that are employed to carry out a task. The resulting service, journal, listens to the network on a specified port for incoming status messages. The messages can be transported using either UDP or TCP and can encode sufficient information so that the end user can identify the host, service instance and context of all messages that arrive. Support for user specified meta data is provided, making it possible for application specific context to be encoded in the message without the need for any extension on the part of the journal service. This is extremely important since it ameliorates the difficulties associated with debugging a distributed application.

Testing of the network status between the machine that manages a distributed computation and the various hosts where services might be deployed is carried out by component ntp. Like all distributed components, ntp consists of a service that is deployed on a remote host and a client process that is deployed on the controlling machine. The latter sends to the former a carefully designed sequence of packets that allow the client to compute an estimate of the instantaneous network bandwidth between the two hosts. This component is implemented by wrapping the

applications provided in freely available package by the same name that has been in use for more than fifteen years.

This method of testing the status of the network has several disadvantages. First, the testing method provided by ntp does not allow an accurate estimate of the latency of the network; it appears that a simple extension of the packet distribution along with better timing information exchange would solve this problem, but such changes are outside the scope of this prototyping effort. Secondly, ntp is unable to construct a predictive model of network behavior and as such, its estimates of network weather are extremely unreliable. A better solution would provide estimates on how likely a certain amount of data is to be transferred between the two hosts in a given amount of time based on both instantaneous and historical performance of the network.

In order to distribute components among the various hosts, we have constructed the gsl package. Package gsl consists of a set of components that facilitate remote launching of components by using only user-space tools based on the ssh family. This allows users to build and deploy distributed applications without requiring special privileges or the involvement of system administrators on the remote machines. Users take advantage whichever machines they have access to, by installing their software on these machines, ensuring that the correct environment is prepared for remote jobs and launching the relevant services.

Data flow between processes, whether they are on different machines or not, turned out to not require the construction of any new components. The machinery constructed to facilitate remote procedure calls is sufficiently powerful to allow the exchange of arbitrary data streams. However, the framework provides support for the construction of specialized transport components that can take advantage of whatever features of the data stream are known in order to optimize performance. Such transport algorithms tend to require advance knowledge about what is being transferred and therefore are best written as part of the optimization step of an application that is already debugged and running correctly.

The finalization of distributed computations is handled by extending the interface of a distributed service. Finalization is now performed when the application instructs the service proxy to clean up after itself, which causes the proxy to send the finalize command to the remote service. The implementation of the finalization behavior is service dependent, so the framework gives services the opportunity to customize the finalization steps. In particular, services are classified as restartable or not, depending on whether each use of the service in a given computation pollutes its internal state to such a degree that the service cannot be used by any other client and must be restarted each time. Both behaviors are available as mix-in classes that allow the developer of a service to specify and then further customize the finalization steps.

These tasks form the minimal set of requirements for a robust distributed system. Even though they are to be considered prototypes and not ready for deployment by general users, they are functional enough to allow users to explore strategies for implementing distributed applications.

ACKNOWLEDGMENT

Eric Gurrola and Paul Rosen performed this work at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the Air Force Research Lab. Michael Aivazis performed this work at Para-Sim Inc in Altadena California.

REFERENCES

- [1] Rosen, P. A., S. A. Hensley, I. R. Joughin, F. K. Li, S. N. Madsen, E. Rodriguez, and R. M. Goldstein, Synthetic aperture radar interferometry, *Proceedings of the IEEE*, vol. 88, no. 3, pp. 333-382, March 2000.
- [2] Burgmann, R. P. A. Rosen, and E. J. Fielding, Synthetic aperture radar interferometry to measure Earth's surface topography and its Deformation, *Annual Reviews Earth Planetary Science*, vol. 28, pp. 169-209, 2000.
- [3] Massonet, D., K. L. Feigl, Radar interferometry and its application to changes in the Earth's surface, *Reviews of Geophysics*, vol. 36, no. 4, pp. 441-500, November 1998.
- [4] <http://www.cacr.caltech.edu/projects/pyre/>