Project Final Report:

Building a Community Infrastructure for Scalable On-Line Performance Analysis Tools around Open/SpeedShop

Project Title and PI names

Project Title: "Building a Community Infrastructure for Scalable On-Line Performance Analysis Tools around Open/SpeedShop"
Internal Project Title: “Component Based Tool Framework (CBTF)”
DOE grant numbers:
- University of Wisconsin: ER25934
- The Krell Institute: ER259335
- University of Maryland: ER25936

Project PI:

James E. Galarowicz
The Krell Institute
1609 Golden Aspen Drive, Suite 101
Ames, Iowa 50010

Partner PIs:

Barton P. Miller
Computer Sciences Department
University of Wisconsin
Madison, WI 53706-1685

Jeffrey K. Hollingsworth
Computer Sciences Department
University of Maryland
College Park, Maryland 20742

Philip Roth
Future Technologies Group
Computer Science and Math Division
Oak Ridge National Laboratory
One Bethel Valley Road
P.O Box 2008, MS-6173
Oak Ridge, TN 37831-6173

Martin Schulz
Center for Applied Scientific Computing (CASC)
Lawrence Livermore National Laboratory
PO Box 808, L-560
Livermore, California 94551

Unfunded Collaborators:

David Montoya, Samuel Gutierrez, Dane Gardner
Los Alamos National Laboratory

Mahesh Rajan
Sandia National Laboratories

Ciera Jaspan
Carnegie Mellon University
Date of the report and period covered by the report .................................................. 2

Executive Summary ........................................................................................................ 2

Comparison of Goals/Objectives and Accomplishments and Project Activities: ................. 3

Project Activities: ............................................................................................................. 3

  Enhancement and implementation of the initial design of the scalable component tool based framework. (The Krell team, in close collaborations with all project partners and unfunded collaborators) .......................................................... 3

  Componentization of Open|SpeedShop (The Krell team) .................................................. 4

  Component Infrastructure Libraries and Status ............................................................... 4

  Integration of MRNet overlay network into the data aggregation component (The Krell team) .......................................................... 5

  Capability Machine Support (The Krell team) ............................................................... 6

  Research into Component Framework Interfaces (The Krell team) ................................. 7

  Produce survey of performance tool needs for at least five applications of interest to the DOE Office of Science (The ORNL team) .......................................................... 8

  Interactions with application teams (The LLNL team) .................................................... 8

  Dyninst binary rewriter developments (The University of Wisconsin team) ....................... 9

  Progress and Accomplishments for entire project with UMD components (The University of Maryland team) .......................................................... 10

Comparison of the actual accomplishments with goals and objectives: ................................ 10

Identification of Products developed under the award and technology transfer activities ........ 11

  Dissemination and Outreach (Entire Project Team) ........................................................ 11

    Talks, Papers, and Tutorials featuring Open|SpeedShop and/or CBTF: ................................ 11

    Software Products (Krell Institute, Argo Navis, LANL) .................................................. 12

    Software Products: Dyninst binary rewriter (The University of Wisconsin team) ................ 13

    Software Products: Active Harmony using CBTF (The University of Maryland team) .......... 13

Date of the report and period covered by the report

Date of Report: December 19, 2013   Period Covered: October 1, 2009 to September 30, 2012

Executive Summary

In this project we created a community tool infrastructure for program development tools targeting Petascale class machines and beyond. This includes tools for performance analysis, debugging, and correctness tools, as well as tuning and optimization frameworks. The developed infrastructure provides a comprehensive and extensible set of individual tool building components. We started with the basic elements necessary across all tools in such an infrastructure followed by a set of generic core modules that allow a comprehensive performance analysis at scale. Further, we developed a methodology and workflow that allows others to add or replace modules, to integrate parts into their own tools, or to customize existing solutions.

In order to form the core modules, we built on the existing Open|SpeedShop infrastructure and decomposed it into individual modules that match the necessary tool components. At the same time, we addressed the challenges found in performance tools for petascale systems in each module. When assembled, this instantiation of community tool infrastructure provides an enhanced version of Open|SpeedShop, which, while completely different in its architecture, provides scalable performance analysis for petascale applications through a familiar interface. This project also built upon and enhances capabilities and reusability of project partner components as specified in the original project proposal.
The overall project team’s work over the project funding cycle was focused on several areas of research, which are described in the following sections. The reminder of this report also highlights related work as well as preliminary work that supported the project.

In addition to the project partners funded by the Office of Science under this grant, the project team included several collaborators who contribute to the overall design of the envisioned tool infrastructure. In particular, the project team worked closely with the other two DOE NNSA laboratories Los Alamos and Sandia leveraging co-funding for Krell by ASC’s Common Computing Environment (CCE) program as laid out in the original proposal. The ASC CCE co-funding, coordinated through LLNL, was for 50% of the total project funding, with the ASC CCE portion of the funding going entirely to Krell, while the ASCR funding itself was split between Krell and the funded partners. This report covers the entire project from both funding sources.

Additionally, the team leveraged the expertise of software engineering researchers from Carnegie Mellon University, who specialize in software framework design, in order to achieve a broadly acceptable component framework. The Component Based Tool Framework (CBTF) software has been released to the community. Information related to the project and the released software can be found on the CBTF wiki page at: http://sourceforge.net/p/cbtf/wiki/Home.

**Comparison of Goals/Objectives and Accomplishments and Project Activities:**

**Project Activities:**

This report describes the progress made in the three years of work under this funding. All project partners collaborated through regular telephone conferences, including a weekly core team telephone conference. Additionally, the project team held several face-to-face meetings, which in most cases were held in conjunction with existing conferences and meetings attended by the project partners to minimize travel expenses. In the following sections we will detailed the accomplished achievements separated into major areas of research.

**Enhancement and implementation of the initial design of the scalable component tool based framework. (The Krell team, in close collaborations with all project partners and unfunded collaborators)**

During the initial phase of this project, the entire project team – under Krell’s leadership – created a concrete and scalable design for the new Component Based Tool Framework (CBTF) based on the performance pipeline concept laid out in the initial project plan. The design was refined over the entire funding cycle as a result of the regularly scheduled telephone conference and during discussions at project face-to-face meetings. This design serves as the foundation for the creation of three primary component based tool framework infrastructure libraries and the initial tool components, services, and messages required to create a tool based on the framework. These libraries and components are described in more detail in the section below titled “Component Infrastructure Libraries and Status”.

The Krell team made project design documents and all meeting minute documents available on the project wiki members site as well as a public project wiki at: http://ft.ornl.gov/doku/cbtfw/start.

The Krell team used the project web site and source control (git) mechanisms for this project, which were created at ORNL by Phil Roth. The project team used the control site as an internal repository, but since the
project funding cycle end, we have opened the project sources and information to the general public and use the CBTF sourceforge web pages as the community portal for the CBTF project: http://sourceforge.net/p/cbtf.

**Componentization of Open|SpeedShop (The Krell team)**

We completed the projects related to the re-factorization of the Open|SpeedShop tool and separated the collector run time and collector client code into separately build-able entities in preparation to use Open|SpeedShop on capability machines, such as the Cray and Blue Gene machines. As a result of this work, Open|SpeedShop now supports both the Cray and BG/P and BG/Q platforms. During the course of the project, these changes were also leveraged as a guide to building CBTF tool services and messages upon which tool builders and users can create their own tools.

**Component Infrastructure Libraries and Status**

In the past year, the Krell Institute team has developed three CBTF infrastructure libraries that form the core component based tool framework support. The three key core libraries are:

- **libcbtf**, which provides component abstraction, component metadata, and component interconnection support.
- **libcbtf-xml**, which provides XML-Based definition of single process component networks
- **libcbtf-mrnet**, which provides XML-Based definition of MRNet-based distributed component networks

The CBTF library named libcbtf enables users to define a component and connect it with other components. For the latter, components are based on a data-flow model, accepting zero or more inputs, performing arbitrary processing, and emitting zero or more outputs. They are written using C++, inheriting from a base class that provides basic infrastructure. They provide metadata in form of the component’s type, version, and input/output names and types. All components are versioned and multiple versions the same component can co-exist. Finally, components can be packaged in a variety of different ways including being embedded inside an executable, as a shared library, and as a runtime plugin. Figure 1 illustrates the concept of a component in the CBTF context where there are zero or more inputs and zero or more outputs for each component.

CBTF individual components can be connected together to form component networks. Each network contains one or more component instances and connections between those component instances. Any topology is possible, from pipelines to graphs with cycles. Networks are recursive in the sense that the network is itself a component, with input and outputs, which can be instantiated and further connection to other networks. Finally, the components to be instantiated (including version ranges), along with the interconnections, are specified via XML files. Figure 2 shows three components with their respective component connections, which
defines a component network. That component network is itself a component with two inputs and one output.

The CBTF library named libcbtf-xml provides the ability of specifying component network connections within a single process via an XML file. A function call, “registerXML (<toolname>.xml)” creates the network for the user. libcbtf-xml uses the low-level library libcbtf to do the connections. There is an XML schema that defines what is a legal specification. With this functionality users can quickly prototype new tools from existing components by only specifying XML text and without any further programming, opening the door for more application specific and end-user controlled tool functionality.

With the ability to create single process component networks, the next step was to create infrastructure support to distribute these single process component networks across the nodes of large-scale systems. We build on top of MRNet, an implementation of a tree based overlay network that enables scalable tool communication and aggregation, and developed the CBTF library libcbtf-mrnet. The idea is similar to the single process XML definition. However, the MRNet tool specification file specifies the component networks across the MRNet overlay network tree. It considers at what level to put the component network in the MRNet tree and how to connect the components on that level. The library remains topology neutral, by passing the topology through to MRNet. Component network specifications are relative to the MRNet backend and the MRNet frontend node level. The tool builder can specify all node levels get a component network or only node levels based on a relative value. A complete component network is located on each node in the MRNet network. These component networks are connected via symbolically named MRNet streams that can move data either up or down the MRNet network. Figure 3 illustrates three levels of an MRNet tree with component networks distributed across those levels. The red lines indicate the MRNet streams through which the data moves upstream or downstream depending on the component or component network’s programming logic.

As with non-distributed component networks, distributed component networks are recursively defined and specified via XML files. MRNet tool networks are also specified through XML specification files, and hence, as with single process tool, a tool builder or user, could create a component network via the base library (libcbtf) or through XML specification files using libcbtf-xml for single process networks or libcbtf-mrnet for distributed networks.

Integration of MRNet overlay network into the data aggregation component (The Krell team)

In the first year of this project we integrated the latest version of MRNet (beta 3.0) into the mainline Open|SpeedShop version as a precursor to including it in the new Component Based Tool Framework source. During the second year of the project, the Krell team, in collaboration with the University of Wisconsin team, updated the MRNet version and used the latest version of MRNet in the creation of the CBTF infrastructure component network distribution library: libcbtf-mrnet.
Starting with the second year, the Krell team worked on using the Lightweight MRNet Backend instantiation mechanism in MRNet to create a performance data file transfer mechanism that would transfer offline performance data directly from the application to the client tool. This research was predicated on the results of the research mentioned above relating to creation of the libcbtf-mrnet library. The benefit of this research is the elimination of thousands of file system writes of the raw performance data gathered by a CBTF tool based on this technology. This is essential to scaling CBTF and for the deployment of O|SS on the upcoming large scale platforms within the DOE laboratories.

In the third year of the project, the Krell team conducted research, development, and integration of filtering capabilities into the CBTF tool components making use of the MRNet overlay network that is part of the CBTF data transport mechanism. We also supported the efforts of project team members to use the CBTF infrastructure to create additional, non O|SS related tools, by providing documentation and direct support to project team members that supported their efforts to use the CBTF infrastructure to create tools and/or additional CBTF components that can be used to create tools. A short tutorial was developed explaining how to add capabilities and leverage existing tool services, messages, and components.

- How to create components
- How to connect components
- How to build services, messages and necessary core services
- How to distribute across MRNet
- How to start up tools

**Capability Machine Support (The Krell team)**

During the first year, we had built a version of the Open|SpeedShop run time environment for the Cray-XT platform. We then modified the existing Open|SpeedShop build configuration system to support targeted builds of static base collectors to support the Cray-XT/XE platforms. We have shown that the targeted Open|SpeedShop Cray-XT/XE version base collectors (pcsamp, usertime, hwc, io, mpi) were built and could be linked into a Cray-XT/XE application. The Cray-XT/XE porting work was completed, and efforts to add scaling improvements are still ongoing. We have demonstrated a working version of the Open|SpeedShop run time environment for the Cray-XT/XE platform and used the targeted Open|SpeedShop Cray-XT/XE version of the collectors to show that the Open|SpeedShop base experiments are successful on the Cray-XT/XE. We ran this version of Open|SpeedShop to analyze the PEPC application using 12,000 processors of ORNL's Cray-XT Jaguar for the performance tool challenge requested by the Dagstuhl 2010 seminar organizers. During those experiments, we demonstrated the scalability and low overhead of the entire infrastructure. Further, these experiments provided the necessary input to the ongoing scaling refinements.

In addition to the static application support, we have added support for dynamic executables which enables Open|SpeedShop on the Cray-XT/XE platforms to be used in the same fashion as the on large-scale clusters by transparently preloading the tool’s data collectors. This greatly reduces the usability of the tools since application developers no longer have to prelink their codes and is another step in closing the gap in the support for capacity and capability machines.

During the second year, we created and demonstrated a working version of the Open|SpeedShop on a Linux PPC/PPC64 platform. We demonstrated that each of the Open|SpeedShop base experiments (pcsamp, usertime, hwc, io, mpi) were functioning correctly by displaying data gathered on the PPC/PPC64 platform.
This work was one of the initial phase’s necessary, prior to the port of Open|SpeedShop to the BG/P platform. During that same funding cycle the Krell team created a version of the Open|SpeedShop that gathers and displays performance information for the Blue Gene platform and used that version to prove that the Open|SpeedShop base experiments work on the BG/P platform. This work was completed using the target machine, Intrepid: the BG/P platform at Argonne National Laboratory. Additional work was required, and done at a later time, to support performance analysis of shared/dynamic applications on the Blue Gene platform.

This work on PPC/PPC64 platforms also formed the basis for a support of the, at the time, upcoming BG/Q platform and its installations at ANL and LLNL. To facilitate this further, the Krell team worked with IBM and the BG/Q teams at ANL and LLNL to ensure that all software requirements needed for CBTF was present.

During the third year, we created and demonstrated a working version of the Open|SpeedShop on a Linux BG/Q platform. We also continued their effort to enhance Open|SpeedShop support for capability machines by expanding the Open|SpeedShop Blue Gene version support to include shared/dynamic application support. This is currently implemented and working on Blue Gene platforms using the SLURM operating system. Work was also done to integrate Blue Gene node personality support into the Blue Gene version of Open|SpeedShop during the third funding cycle. Work on Cobalt job control based Blue Gene systems is pending. We also refined the Cray platform support to include the Cray-XK platform and simplified the build process.

We also reached a higher processor count for our runs on capability platforms during the third year of the project where we were able to do performance analysis on applications running with 50,000 processors.

Research Into Component Framework Interfaces (The Krell team)

In the first year of the project, we researched a number of possible existing component based framework options but found that none of the various options examined met the needs of the project. The first fundamental choice in designing the new framework was whether to go with an interface based model, or a data flow based model. In our research, a widely accepted technology for doing data-flow oriented, C++, component on Linux was not found. Our work is based on the notion of “flow-based programming”.

During the second year of the project, we implemented, enhanced and added additional capabilities to a component based tool framework infrastructure based on the research done during the first year of the contract. This is discussed in detail in the “Component Infrastructure Libraries and Status” section above. In all the above research areas the Krell team continued actively seeking feedback and alternative approaches from the project partners. Email interactions as well as face-to-face meetings were held throughout the funding period and were essential to integrate all components and use cases targeted in this project. Further, the design was presented to the community on several occasions, introducing the concept of and gathering feedback on CBTF.

In the third year of the project, we extended the CBTF component network concept to include a transport mechanism based on MRNet and creating communication node filters from CBTF component networks. We conducted research, development, and integration of filtering capabilities into the CBTF tool components making use of the MRNet overlay network that is part of the CBTF data transport mechanism. We also supported the efforts of project team members to use the CBTF infrastructure to create tools by providing documentation and direct support to project team members that supported their efforts to use the CBTF infrastructure to create tools and/or additional CBTF components that can be used to create tools.
During the last funding cycle, we expanded the new Component Based Tool Framework. This expansion was aimed at the transport mechanism and components to allow alternative uses of the transport mechanism, including Dyninst instrumentation and continued enhancement of offline data transport.

The accomplishments for the enhancement of the base CBTF infrastructure were:

- Added additional message, services, and core facilities to the CBTF tool set that will allow tool builders and users to gather hardware counter data and call path-profiling information using CBTF tool components.
- Researched tool start-up mechanisms for CBTF based tools
  - Research existing start-up tools and libraries such as LIBI and launchmon to support CBTF tool start-up and launch.
  - Research using dynamic instrumentation, such as Dyninst from the Universities of Wisconsin and Maryland as a means to inject code to aid in tool start-up.
- Added support into the CBTF infrastructure which allows CBTF base tools to use the MRNet Lightweight Backend attach mode

Additionally, we followed the developments for BG/Q in close collaboration with IBM, LLNL, and ANL to enable an early port of CBTF to this architecture. However, the MRNet Lightweight backend support was not available in time to be used for this purpose, during this project's funding cycle.

*Produce survey of performance tool needs for at least five applications of interest to the DOE Office of Science (The ORNL team)*

Over the course of the project, the Oak Ridge National Laboratory (ORNL) team contributed to the project in two areas: applications expertise and web-based services support. With regard to applications, along with other project team members we have begun investigating the use of the existing Open|SpeedShop with applications such as the Community Climate System Model (CCSM) on systems deployed at ORNL. The full CCSM is a highly complex program, combining individual climate models for ocean, atmosphere, land, and sea ice into a single MPI-based program. Because of its complexity and because of several open research questions regarding how to map the program’s individual models to system resources across space and time dimensions, CCSM remains an excellent, challenging test case for the existing Open|SpeedShop and for future tools based on the project’s Component Based Tool Framework (CBTF). In response to recent rapid development on the CBTF, we investigated how to build application-specific tool pipelines using CBTF. With regard to web-based services, the ORNL team provided several services for the project, including a wiki collaboration area and source code repository hosting. In the last year of the project, we modified the wiki to expose some of the content for public dissemination while keeping other areas private for team members.

*Interactions with application teams (The LLNL team)*

Complementing and contributing to the application task lead by ORNL, the LLNL team worked with several code and application teams based at LLNL, which cover the mission space for ASCR and NNSA. In particular, this included working with the hypre team, which provides a solver library used in several application codes relevant to the Office of Science. This provided the base data for several performance evaluations at scale as well as for performance modeling activities. Further, the LLNL team worked with several user groups to
increase the usage of OSS throughout LLNL. We use these efforts to help our scientists understand the performance of their codes and libraries, but also to gather requirements and usage information on Open|SpeedShop to help drive the design of the CBTF. In particular, we continue our focus on using Open|SpeedShop on threaded codes.

The LLNL team contributed several system-level infrastructure components (which are developed separately developed and maintained at LLNL), incl. LaunchMon, P^nMPI, and stackwalker API (which is now maintained in close collaboration between LLNL and the team at UW) and provided support for their integration into CBTF.

Additionally, the LLNL team was responsible for coordinating the collaboration between the partners funded by the Office of Science and NNSA, who co-funds Krell’s development effort. This allows us to leverage the expertise and machine resources between ANL and LLNL on the Blue Gene architectures (for intrepid and dawn) and ORNL and LANL (for Jaguar and Cielo).

**Dyninst binary rewriter developments (The University of Wisconsin team)**

The Dyninst binary rewriter instruments a binary executable file to produce a new, instrumented binary. The binary rewriter enables the user to instrument once and run many times, as opposed to dynamic instrumentation, which instruments the program each time that it is runs. Also, the rewritten binary does not have to be compiled or linked again. The binary rewriter operates on unmodified binaries and does not require debug and symbol information or linker relocation data. This flexibility enables it to work on a wide range of binaries. The Dyninst rewriter can add new library dependencies, so will support instrumentation code that has calls to functions shared objects (dynamically linked libraries) or data references to variables in shared objects.

The binary rewriter parses object files, modifies them and writes a new object file. As of this period, Dyninst fully supports the parsing and writing binaries in the Executable and Linkable Format (ELF), which is used by systems such as Linux and FreeBSD. We support dynamically linked executables, shared libraries and statically linked executables in ELF on Linux/x86, Linux/x86_64, FreeBSD/x86, FreeBSD/x86_64, Linux/PowerPC32 and BG/P platforms.

In the last year, we added binary rewriting support to the FreeBSD and the BG/P platforms. With FreeBSD, most of our existing work on Linux/x86 and Linux/x86_64 translated to this platform. However, we needed to handle a few minor differences between these two operating systems. Similarly, BG/P is derived from Linux/PowerPC32, and hence, most of our existing work for rewriting dynamically linked binaries translated to this platform. However, our two major challenges with supporting the BG/P platform were to parse binary codes produced by the Blue Gene compiler (bgxlc) correctly and to produce rewritten binaries that load and execute correctly in BG/P’s more restrictive operating system environment.

We added support for rewriting statically linked binaries for PowerPC/32 and BG/P platforms. For the most part, this implementation is shared with the x86 platform. However, we had to account for differences between the platforms like architecture specific relocation entry types. Since statically linked executables are more prevalent on BG/P systems, being able to rewrite statically linked binaries will improve the range of applications of Dyninst on this platform.
Progress and Accomplishments for entire project with UMD components (The University of Maryland team)

UMD’s effort as part of this project is to integrate the Active Harmony auto-tuning framework into the Component Based Tool Framework (CBTF) that Krell is leading the development of. We have designed how Active Harmony will utilize the framework, which in particular included the ability to feed information back from the tool to the application and/or system, which is an essential requirement not present in our other, analysis oriented use cases. Using CBTF Active Harmony can overcome it current limitations of needing TCP/IP connections between application process and the Harmony server and eliminate the performance bottleneck of having all the application processes connect directly to the server.

Primarily, the UMD team worked with the rest of the partners to ensure that the tool framework was able to support Harmony’s somewhat unique requirements. In particular Harmony needs to both receive performance data, and send out communications to re-configure the applications. This two-way flow of information is different than traditional performance tools that act as more passive receivers of data from applications.

During the project funding cycle we made progress on preparing the Active Harmony code base for CBTF. In particular, we have re-factored the Active Harmony code to better isolate the communications with the application processes. We worked on a change to the GUI interface system to allow plugging in CBTF components there.

Actual testing with CBTF code happened during the summer of 2012 (i.e. during the third year of the contract). We had to wait until the next code drop from the core CBTF team to have the required functionality in the framework.

The primary technical person on the project for UMD is research programmer Ray Chen. Ray also participated in the design conversations about the CBTF.

Comparison of the actual accomplishments with goals and objectives:

The original goals were to design and implement a general and reusable infrastructure targeted for petascale systems by designing, implementing, and evaluating a general, flexible tool infrastructure supporting the construction of performance tools as "pipelines" of high-quality tool building blocks, which we call components. These tool building blocks provide common performance tool functionality, and are designed for scalability, lightweight data acquisition and analysis, and interoperability. For this project, we build on Open|SpeedShop, a modular and extensible open source performance analysis tool set. All components need to be designed for scale, a task made more difficult by the need to provide general modules. The infrastructure must support online data aggregation in order to cope with the expected large amounts of data. Petascale machines are likely to provide limited OS capabilities requiring new and lightweight data acquisition techniques. We must be able to efficiently store the performance data using novel, distributed performance data storage techniques. We must be able to map any combination of tool components to each of the target architectures. And we need to design interoperable tool APIs and workflows that are concrete enough to support the required functionality, yet provide the necessary flexibility to address a wide range of tools.
The fundamental goals and objectives of the project were met by our accomplishments as laid out in this report. As stated above in the technical sections, we developed a component based tool framework which allows rapid prototyping of scalable tools and have recreated the Open|SpeedShop performance tool using the CBTF components. Other example tools and the tools built at Los Alamos National Laboratory have been based on CBTF and show the scalability and flexibility promise that were two of the project goals.

**Identification of Products developed under the award and technology transfer activities**

**Dissemination and Outreach (Entire Project Team)**

The entire project team actively engages in disseminating the project results to the academic community, in particular other tools researchers, and potential users. These activities are essential to establish broad support for the new framework and to allow other researchers to participate in this project, which is ultimately intended as a community approach. At the same time, it is important to engage users at and beyond our laboratories and departments, to gather early feedback and to establish a user base. Since the focus was so far on designing and implementing the base infrastructure, these efforts were focused on presentations and tutorials to reach a wide audience within the tools community and to gather feedback from other tools groups. With CBTF becoming available as a stable prototype platform, it provided additional publishing opportunities for both the infrastructure and tools built on top of it, which we targeted for the last funding cycle of this project and beyond. Below are some highlights from these activities:

- CScADS Meetings: (August 2010-2012) – all project partners will participated in this DOE/OASCR supported events and we used this forum to present the CBTF to the tools community and to gather feedback on its design.

**Talks, Papers, and Tutorials featuring Open|SpeedShop and/or CBTF:**


- Jim Galarowicz, David Montoya “CBTF Design” CSCADS Meeting, Snowbird, Utah. July 2010 – all project partners will participate in this DOE/OASCR supported event and we used this forum to present the CBTF to the tools community and to gather feedback on its design.


Software Products (Krell Institute, Argo Navis, LANL)
The Component Based Tool Framework (CBTF) project created a community tool infrastructure that provides individual tool building components to allow others to add or replace modules, to integrate parts into their own tools, or to customize existing solutions. In other words, the CBTF project provides support for creating a compatible interface for components, finding compatible components and defining an interface API that facilitates communication between components. In the CBTF framework, a component is defined as a “black box” entity with zero or more inputs and zero or more outputs. You may connect these components into component networks both distributed and non-distributed. The CBTF transport mechanism, based on MRNet, allows distributed tools to automatically become highly scalable. The Component Based Tools Framework (CBTF) version 1.1 release is available on sourceforge.

To obtain the CBTF project sources these are the four CBTF based project pages, which point to the, open source, source repositories:

CBTF Project Page
Project Page: Krell Institute Contributions to CBTF
Project Page: Argo Navis Contributions to CBTF
Project Page: LANL Contributions to CBTF
CBTF project contains the base code for CBTF that supports creating components, component networks and the support to connect these components and component networks into sequential and distributed network tools.

CBTF Krell project contains the Krell Institute contributions to the CBTF project. These contributions include many performance data collectors and support libraries as well as some example tools that drive the data collection at HPC levels of scale.

CBTF Argo Navis project contains the CUDA collector and supporting libraries, that was done as a result of a DOE SBIR grant to Argo Navis Technologies, LLC, based on the same infrastructure developed in the ASCR funded project.

CBTF LANL project contains a memory tool and a data center system-command, monitoring tool.

**Software Products: Dyninst binary rewriter (The University of Wisconsin team)**

The Dyninst binary rewriter instruments a binary executable file to produce a new, instrumented binary. The binary rewriter enables the user to instrument once and run many times, as opposed to dynamic instrumentation, which instruments the program each time that is runs. The binary rewriter is now available in the Dyninst official release.

**Software Products: Active Harmony using CBTF (The University of Maryland team)**

We prepared the Active Harmony code base for CBTF. In particular, we have re-factored the Active Harmony code to better isolate the communications with the application processes and changed to the GUI interface system to allow plugging in CBTF components there. A version of Active Harmony using CBTF is available on request. The re-factored Active Harmony system has been incorporated into the latest release of the software and is available for download from www.dyninst.org/harmony.