FICUS - A Federated Service-Oriented File Transfer Framework

Adam Turner[1] and Michael Sobolewski[1]

SORCER Research Group, Texas Tech University, Lubbock, Texas, USA.

Abstract. The engineering data of a large enterprise is typically distributed over a wide area and archived in a variety of file systems and databases. Access to such information is crucial to team members and relevant processing services (applications, tools and utilities) in a concurrent engineering setting. However, this is not easy because there is no simple way to efficiently access the information without being knowledgeable about various file systems, file servers, and networks, especially when complex domain related data files get bigger. In a concurrent engineering environment, there is every need to be aware of the transparent and dynamic data perspectives of the other members of the team.

We have developed a Federated Service-Oriented File Transfer Framework called FICUS (Files In Chunks Utilizing Storage) with the objective to form dynamic federations of network services that provide engineering data, applications and tools on a grid. This framework fits the SORCER philosophy of grid interactive service-oriented programming, where users create distributed metaprograms using federated providers along with FICUS repository providers.

Our paper describes the methodology of how FICUS works along with the details of the implementation and extensions planned for the future. We believe the performance and reliability offered by FICUS will make it a very useful distributed file transfer protocol for a large design team and will make it very convenient to integrate heterogeneous legacy file systems.

Keywords. Data sharing, distributed file systems, federated systems, collaborative work.

1 Introduction

Managing engineering data is becoming an increasingly complex task. Heterogeneity of hardware and software platforms is one of the barriers to be overcome in achieving this end. With increasing use of computers, we have islands of automation that have resulted in information archival in legacy file systems.

[1] SORCER Research Group, Computer Science Dept., Texas Tech University, Box 43104, Boston & 8th, Lubbock, TX 79409, USA; Tel: +1-806-742-5851; Email: sorcer@cs.ttu.edu; http://sorcer.cs.ttu.edu

This makes the access to information easy for someone who uses a single repository as their primary or native environment. However, the information access problem increases manyfold when one wishes to access enterprise-wide information. Access to enterprise-wide information is very important in a collaborative setting when a number of people access a corpus of information, albeit with different perspectives. Major problems to be addressed include how to integrate all the information, how to deal with legacy systems and how to provide a wide view to the user abstracting all the hardware and software specifics of the individual systems. The federated service-oriented file transfer framework (FICUS) we describe in this paper enables access to distributed and replicated information over a wide area with special emphasis on efficient access to data repositories. This includes CAD drawings and other kinds of raster and vector data, in addition to voice and video clips that will be archived in the future. The need for accessing distributed information by people viewing from different perspectives arises in a concurrent engineering setting. Also, the efficient file download by many services sharing the same data becomes essential when requestors share the same copy of a master file at the same time.

Building on the OO paradigm is the service-object-oriented (SOO) paradigm, in which the objects are distributed, or more precisely they are remote (network) objects that play some predefined roles. A service provider is an object that accepts remote messages, called service exertions, from service requestors to execute an item of work. A task exertion is an elementary service request – a kind of elementary remote instruction (statement) executed by a service provider. A composite exertion, called a job exertion, is defined in terms of tasks and other jobs - a kind of procedure executed by a service provider. The executing exertion is a SOO program that is dynamically bound to all relevant and currently available service providers on the network. This collection of providers identified in runtime is called an exertion federation, or an exertion space. While this sounds similar to the OO paradigm, it really isn't. In the OO paradigm, the object space is a program itself; here the exertion space is the execution environment for the exertion, which is a network OO program. This changes the game completely. In the former case, the object space is hosted by a single computer, but in the latter case the service providers are hosted by the network of computers. The overlay network of service providers is called the service provider grid [5-7, 9] and an exertion federation is called a virtual metacomputer. The metainstruction set of the metacomputer consists of the method set defined by all service providers in the grid. Do you remember the eight fallacies of network computing? Creating and executing a SO program in terms of metainstructions requires a completely different approach than creating a regular OO program. In other words, we apply in FICUS the OO concepts directly to the service provider grid.

The SORCER environment [1, 10, 14-16, 20, 22-28] provides the means to create interactive SOO programs and execute them without writing a line of source code via zero-install, interactive service interfaces. Exertions can be created using interactive user interfaces downloaded directly from service providers, allowing the user to execute and monitor the execution of exertions in the SOO metacomputer. The exertions can also be persisted for later reuse. This feature allows the user to quickly create new applications or programs on the fly in terms

of existing tasks and jobs. SORCER introduces federated method invocation based on peer-to-peer (P2P) [17, 18] and dynamic service-oriented Jini technology [4, 8, 12, 13, 18, 19, 21].

To integrate applications and tools on a B2B grid with shared engineering data, the File Store Service (FSS) [5] was developed as a core service in FIPER/SORCER. The value of FSS is enhanced when both web-based user agents and service providers can readily share the content in a seamless fashion. The FSS framework fits the SORCER philosophy of grid interactive SOO programming, where users create distributed programs using exclusively interactive user agents. However FFS does not provide the S2S flexibility with separate specialized and collaborating service providers for file storage, replication, and meta information that have been added in the SILENUS federated file system [1].

In this paper, the FICUS federated service-oriented file transfer framework is described that allows an exertion federation for collaborative, efficient data sharing across federating service providers in terms of files split into smaller chunks that are replicated and stored at multiple locations.

2 FICUS Architecture

FICUS has been designed to explore the file sharing concepts used in modern peer-to-peer technologies such as BitTorrent [2, 3], and investigates how they can be applied to a file system. FICUS is an extension to SILENUS, a federated file system developed at Texas Tech University [1]. The SILENUS file system is comprised of several network services that run within the SORCER environment, each of which provides a functional aspect of the file system. These services include a byte store service for holding file data, a metadata service for holding metadata information about the files (such as file names), several optional optimizer services, and façade services to assist in using these services. SILENUS is designed so that many instances of these services can run on a network, and the required services will federate together to perform the necessary functions of a file system. FICUS adds support for storing very large files within the SILENUS file system by providing two more services: a splitter service and a tracker service. When a file is uploaded to the file system, the splitter service determines how that file should be stored. If a file size is above a predetermined threshold, the file will be split into multiple parts, or chunks, and stored across many byte store services. Once the upload is complete, a tracker service keeps a record of where each chunk was stored. When a user requests to download the full file later on, the tracker service can be queried to determine the location of each chunk and the file can be reassembled.

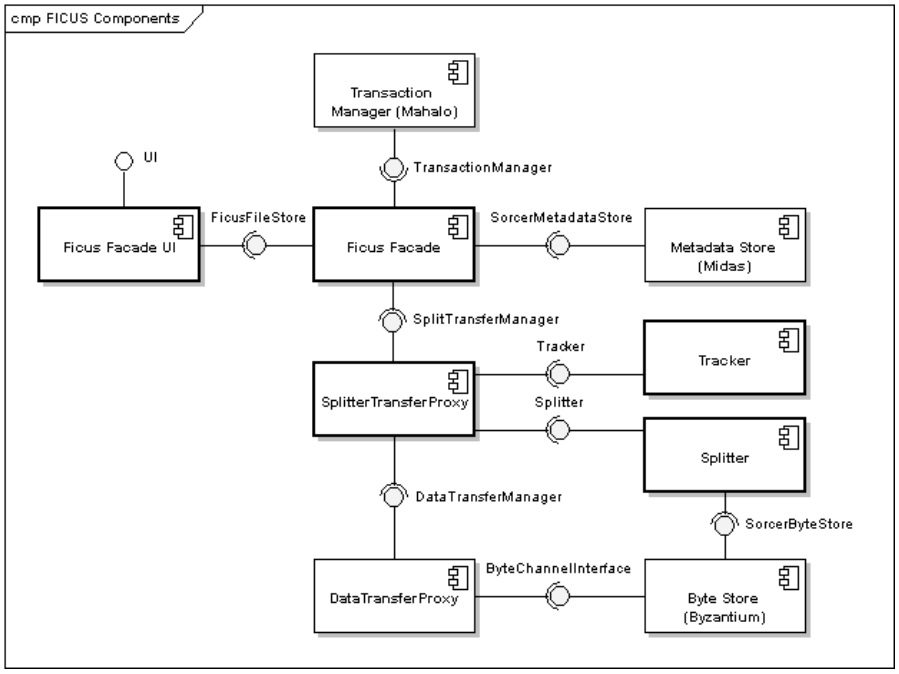

Figure 1. FICUS component diagram

2.1 Splitter Service

BitTorrent [2, 3] doesn't really have any set rules for determining file splitting parameters other than that all pieces must be of the same size (except for possibly the last piece) and the piece size in bytes must be a power of 2. It is normally left up to the hosting user to determine what the piece size should be used. In a file system, this decision should be handled automatically and transparently and should not bother end users with the specifics. Thus, a splitter service is responsible for determining whether or not a file should be split, and if so, what parameters should be used for splitting. Administrators can affect many of the parameters used in making this decision in order to optimize network, storage, and file usage. For example, an administrator can specify the minimum file size required for a file to be considered for splitting. A minimum and maximum chunk size can be specified along with a minimum and maximum number of file splits to use for any file. The splitter can use this information to calculate the optimum chunk size to use for a file based on the file's size and possibly other parameters as well, such as available storage space on each of the byte store services.

The splitter also provides services for splitting and reassembling large files on the requestor side through the use of proxies. A splitter proxy object can concurrently manage multiple byte store proxy objects for communicating directly with various byte store services. When a user or other agent uploads a large file, the splitter proxy can send parts of the file simultaneously to individual byte store

services to store as chunk files. A splitter proxy could then download these chunk files from the multiple byte stores simultaneously and save them as file segments to recreate a copy of the original file.

2.2 Tracker Service

BitTorrent uses a tracker to help peers discover each other and to help peers determine the location of desired file pieces. A tracker service for FICUS provides similar functionality. When a large file is uploaded in chunks to the file system, the location of each chunk is recorded and this information is given to the tracker for storage in a database. During replication, a replicator service can also notify the tracker of any new chunk files that have been made. In addition to chunk locations, the tracker also records the size of the original file, the chunk size used, how many chunks a file has been split into, and an optional checksum for each chunk to see if the chunk file has been corrupted. When a split file needs to be retrieved, the tracker can be queried to find the locations of each chunk needed to completely reassemble the file.

Since a tracker service handles location information for files, it acts as a logical extension to a metadata store service. When a file is stored on a byte store service, the byte store names the file with a UUID to provide unique and persistent identification for the file content. The metadata store normally keeps track of where a file is located using the service ID of the byte store upon which a file is stored along with the file's UUID. In the case of split files, a tracker service records this information for each chunk file while the metadata store records the service ID of any tracker services that are tracking the file along with a similar UUID with a numeric extension to refer to the record number of the file within the tracker.

2.3 Façade Service

With the inclusion of these new services, an updated façade service is needed to assist with the coordination of the various file system services. Specifically, the façade service is responsible for discovering the metadata store services needed for browsing through the file system, and splitter services to initiate file storage and retrieval. When uploading files or performing other operations that require data alterations on more than one file system service, the façade is also responsible for keeping these actions under transactional semantics to verify that all required operations either complete successfully or abort. Due to the nature of the façade service, it can act as an entry point for other services into the file system. Through the use of a service browser such as Inca X [11], a user could obtain a GUI (ServiceUI [21]) for a façade service and interact with the file system directly without the need to install any additional software. The façade could also be used to manage and distribute remote event notifications. For example, once an upload has completed, a façade service could notify a replication service with the appropriate information to begin replicating the new file to additional storage services.

3 Uploading and Downloading

Many of the concepts found within FICUS can be compared to those found in other Internet based peer-to-peer programs such as BitTorrent. For example, in the case of BitTorrent, users can typically use their favorite web browser to find and download relevant torrent files located on public web servers. Similarly, a FICUS façade service can help users locate files they want from the file system using file records found on metadata store services. BitTorrent uses torrent files that contain meta-information about a file or set of files along with a URL for an Internet based tracker service used for connecting to peers. In FICUS, the metadata entry for a requested file contains a reference to a FICUS tracker service to use, which holds additional metadata information about the file along with file chunk locations. A BitTorrent client application can be used to acquire and share pieces of a file with peers. Similar functionality is handled by a proxy object provided by a FICUS splitter service. The following sections provide more information about how files are uploaded and downloaded using BitTorrent and FICUS to provide a better understanding of how the two compare.

3.1 BitTorrent

BitTorrent is a peer-to-peer technology that has become quite popular over the past few years. It allows a person to share a large file (or set of files) with many users while transferring relatively little data. It accomplishes this task by breaking the file into smaller pieces that can be quickly shared between other users. Once a user has downloaded a file piece, it can send the completed piece to other users who still require it. In essence, the upload bandwidth required to share the file is now distributed amongst the peers rather than making a single server solely responsible for doing all the uploading. User systems within the peer "swarm" are able to find each other and figure out which systems have desired pieces through the use of a tracker service running on the Internet.

In order to begin serving a file, a user would go through the following steps.

1. Find a tracker to manage the peers involved in transferring file pieces.
2. Generate a metainfo (torrent) file using the complete file to be served and the URL of the tracker.
3. Upload the torrent file to a website.
4. Start a BitTorrent client using the torrent file to begin seeding the full file.

Downloading the full file involves the following steps.

1. The user finds and downloads the torrent file from the web server.
2. The user loads the torrent file into their BitTorrent client.
3. The BitTorrent client connects to the tracker to find other peers.
4. The BitTorrent client downloads pieces of the file from others and shares the pieces it has with others.

Once all pieces are distributed into the peer swarm, clients can share with each other until they all have the full file.

3.2 FICUS

FICUS uses a methodology similar to BitTorrent for uploading and downloading files. However, since FICUS is a file system rather than simply an Internet file sharing service, there are some differences between these methods.

In order to upload a file to a FICUS file system, the following occurs.

1. A user requests to upload a file.
2. A façade service forwards file parameters to a splitter service to determine how to handle the file.
3. If the file is large, the splitter will get a reference to a tracker service and send back a splitter transfer proxy.
4. This proxy is sent back to the user end and is given a reference to the file.
5. The proxy sends pieces of the file to byte store services to store as chunks.
6. The location of each uploaded chunk is recorded by the tracker.
7. The tracker's service ID along with the file's record number are given to a metadata store service to indicate where the file locations can be found for future retrieval.

In order to download a file from a FICUS file system, the following occurs.

1. A user requests to download a file.
2. A façade service queries the location of the file from a metadata store.
3. If the file is split, it is forwarded to a tracker.
4. The tracker provides the locations of each chunk.
5. The chunks are downloaded from various byte store services and assembled into the full file.

4 Conclusions

FICUS is able to provide several benefits over traditional client-server file system in which file's are stored in their entirety. Many of these benefits stem from the service-to-service oriented nature of SORCER along with the file splitting capabilities of FICUS. In a traditional client-server based network file system, a large amount of storage space must either be found or created in order to store large amounts of data, especially if the data is contained in only a single file. The speed at which this data can be provided to others is usually limited by the maximum bandwidth available to the server, which can cause severe bottlenecks if many clients request the same data at once. If an error occurs during a file transfer, the client often has to restart the transfer from the beginning, which wastes time and magnifies the bottleneck issue. If the file server goes down, then these files are typically unavailable until the server can be restored. Basically, the major disadvantage of storing whole files within a client-server type file systems is that the server can easily become a single point of failure. To help alleviate this problem, many file servers run on expensive, high end, redundant server equipment. High speed RAID arrays are often employed to not only help recover from a hard drive failure, but also to provide increased throughput for client requests. Additionally, servers are often placed on high speed network segments to handle the necessary bandwidth requirements.

Many of these problems can be avoided by splitting large files into chunks and by using a service-to-service type architecture as provided by SORCER. By storing files in chunks across multiple storage locations, storage and network requirements become much more distributed. For example, when storing a large file, it is usually much easier to find several storage locations with lesser amounts of free space than it is to find a single location with a massive amount of free space. When downloading files, there may be several different storage locations that have the requested data rather than just a single server, thus it is far less likely for any single storage location to become a bottleneck due to a high number of file requests. If files are spread across multiple locations in chunks, a client could download multiple chunks simultaneously, thereby using the aggregate bandwidth of all storage nodes rather than the available bandwidth of just a single server. If an error occurs during a file transfer, then only the erroneous chunk would have to be transferred again rather than the full file, which can save a lot of time and waste less bandwidth.

The concepts and methodologies proposed by FICUS provide other opportunities for enhancing a file system as well. For example, after making a modification to a large file, it may be possible to save only the relevant chunk files that have changed rather than the entire file. This technique has the potential to save drastic amounts of bandwidth and storage space, especially when storing multiple versions of the same file. Special optimizer services could be designed for replicating chunk files that are used most often to more stable and higher powered machines, thus providing greater availability for frequently accessed data. Overall, the concepts proposed by FICUS provide many different avenues for exploration to enhance the scalability, reliability, and performance of distributed network file systems.

References

[1] Berger M, Sobolewski M. SILENUS – A Federated Service-oriented Approach to Distributed File Systems. In: Next Generation Concurrent Engineering. New York: ISPE, Inc., 2005; 89-96.

[2] BitTorrent.org – BitTorrent Protocol Specification. Available at: <http://www.bittorrent.org/protocol.html>. Accessed on: Mar. 8th 2007.

[3] Brian's BitTorrent FAQ and Guide. Available at: <http://www.dessent.net/btfaq/>. Accessed on: Mar. 8th 2007.

[4] Edwards WK. Core JINI Second Edition. (2000). Prentice Hall, 2000.

[5] Foster I. The Grid: A New Infrastructure for 21st Century Science. In: Physics Today. American Institute of Physics, 2002; 55(2): 42-47.

[6] Foster I, Kesselman C, Nick J. Tuecke S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG. Global Grid Forum, June 22nd 2002.

[7] Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001; 15(3).

[8] Freeman E, Hopfer S, Arnold K. JavaSpaces™ Principles, Patterns, and Practice. Addison-Wesley, 1999.

[9] The Globus Project. Available at: <http://www.globus.org>. Accessed on: Mar. 8th 2007.

[10] Goel S, Talya S, Sobolewski M. Preliminary Design Using Distributed Service-based Computing. In: Next Generation Concurrent Engineering. New York: ISPE, Inc., 2005; 113-120.
[11] Inca X™ Service Browser for Jini Technology. Available at: <http://www.incax.com/service-browser.htm>. Accessed on: Mar. 8th 2007.
[12] Jini Architecture Specification. Available at: <http://www.sun.com/jini/specs/jini1_1.pdf>. Accessed on: Mar. 8th 2007.
[13] Jini.org. Available at: <http://www.jini.org/>. Accessed on: Mar. 8th 2007.
[14] Khurana V, Berger M, Sobolewski M. A Federated Grid Environment with Replication Services, In: Next Generation Concurrent Engineering. New York: ISPE, Inc., 2005; 97-103.
[15] Kolonay RM, Sobolewski M, Tappeta R, Paradis M, Burton S. Network-Centric MAO Environment. The Society for Modeling and Simulation International, 2002 Western Multiconference, San Antonio, Texas.
[16] Lapinski M, Sobolewski M. Managing Notifications in a Federated S2S Environment. In: International Journal of Concurrent Engineering: Research & Applications, 2003; 11:17-25.
[17] Li Sing. JXTA Peer-to-Peer Computing with Java. Wrox Press Ltd., 2001.
[18] Oram A (ed). Peer-to-Peer: Harnessing the Benefits of Disruptive Technology. O'Reilly, 2001.
[19] Project Rio: A Dynamic Service Architecture for Distributed Applications. Available at: <https://rio.dev.java.net/>. Accessed on: Mar. 8th 2007.
[20] Röhl PJ, Kolonay RM, Irani RK, Sobolewski M, Kao K. A Federated Intelligent Product Environment. AIAA-2000-4902, 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization. Long Beach, CA, Sept. 6-8th 2000.
[21] The ServiceUI Project. Available at: <http://www.artima.com/jini/serviceui/index.html>. Accessed on: March 8th 2007.
[22] Sobolewski M. Federated P2P Services in CE Environments. In: Advances in Concurrent Engineering. A.A. Balkema Publishers, 2002; 13-22.
[23] Sobolewski M. FIPER: The Federated S2S Environment. JavaOne, Sun's 2002 Worldwide Java Developer Conference. San Francisco, California, 2002. Available at: <http://sorcer.cs.ttu.edu/publications/papers/2420.pdf>.
[24] Sobolewski M, Kolonay R. Federated Grid Computing with Interactive Service-oriented Programming. International Journal of Concurrent Engineering: Research & Applications, 2006; 14(1):55-66.
[25] Sobolewski M, Soorianarayanan S, Malladi-Venkata RK. Service-Oriented File Sharing. Proceedings of the IASTED Intl. Conference on Communications, Internet, and Information Technology, Nov. 17-19th 2003. Scottsdale, AZ. ACTA Press, 2003; 633-639.
[26] Soorianarayanan S, Sobolewski M. Monitoring Federated Services in CE. Concurrent Engineering: The Worldwide Engineering Grid. Tsinghua Press and Springer Verlag, 2004; 89-95.
[27] SORCER Research Group. Available at: <http://www.sorcer.cs.ttu.edu>. Accessed on: Mar. 8th 2007.
[28] Zhao S, Sobolewski M. Context Model Sharing in the FIPER Environment. Proc. of the 8th Int. Conference on Concurrent Engineering: Research and Applications, Anaheim, CA, 2001.