

GridFrame- A Framework for Building Quality Aware Component-based Grid Systems*

PRADEEP J. MYSORE, RAJEEV R. RAJE

Department of Computer and Information Science, Indiana
University Purdue University Indianapolis, 723 W. Michigan
St, SL 280H, Indianapolis, IN 46202, USA.
{pmysore, rraje}@cs.iupui.edu

PURUSHOTHAM V. BANGALORE,

BARRETT R. BRYANT

Department of Computer and Information Sciences,
University of Alabama at Birmingham
Birmingham, AL 35294 U.S.A.
{puri, bryant}@cis.uab.edu

Abstract: Predominantly, the Grid world has focused on discovering and using hardware solutions for executing scientific and mainstream applications. The applications for Grid are typically handcrafted and also assume the presence of an expert user, thus, making this process error-prone. This paper presents a framework, called GridFrame, whose vision is to reduce the complexity of the applications for Grid systems. GridFrame achieves this goal by providing an approach for semi-automatically discovering independently developed components and constructing quality-aware Grid applications using these components.

I. INTRODUCTION

Software Component Frameworks [1] have been established as a standardized way of building commercial distributed applications from independently developed sub-units. However, the Grid world, which is predominantly scientific, has been slow in embracing these concepts [1, 2]. With increasing mainstream usage of Grid Computing, software component composition and reuse through service oriented Grids have become an increasing need for current and future Grid projects.

Despite the popularity of service oriented Grid, several interesting challenges, such as creating applications from pre-existing components, masking their heterogeneity, and reducing the manual involvement in the development phase, have yet to be tackled adequately. The current software development process for Grid applications consists of an initial application development, testing, and validation that is done on local resources with a subset of the program. After the initial validation, these applications are migrated to larger systems. All the necessary pieces for integration are handcrafted and weaved manually to achieve a software realization of any application, thus, achieving little reuse. Also, there is no mature application development environment for the Grid - ad hoc approaches that are prevalent in the high-performance computing domain are used to develop Grid applications. Although, there are a few tools such as graphical modelers [3] that are available for aiding this process, it still requires a significant amount of manual intervention.

This paper describes a framework, GridFrame, for the creation and composition of distributed Grid components. Using GridFrame, programmers can reason about quality of service (QoS) for individual Grid services as well as a constructed Distributed Computing System (DCS) out of these Grid services. The ability to compose and deploy grid-enabled applications from pre-existing components will enable the rapid design and development of next generation distributed applications while promoting better software reuse with the creation of domain-specific component repositories.

II. RELATED APPROACHES

A *Grid experience* is defined as the Grid utilization process, which runs from the Grid application creation to the final deployment and execution of the application. Most of the existing approaches, such as [4, 5] are targeted at the latter, i.e., deployment and execution of the application, and tackle challenges such as requirements analysis, selecting hardware resources and providing middleware facilities for enabling a user to deploy a Grid application(s). Typically, these approaches assume that the Grid application has already been designed and pre-customized to the Grid deployment phase. Only a few approaches (e.g., [3, 6, 7, and 8]) address the challenges of providing a software component framework for creating component-based Grid applications using pre-built components.

Before elaborating on any of the current Grid software component framework approaches, it is necessary to first identify the requirements and constraints that the Grid places on such a framework. In the current Grid scenario (and for purposes of this paper), components are defined to be Open Grid Services Architecture (OGSA) [2] Services deployed in OGSA containers with associated service data indicating their characteristics. These components are characterized by their dynamic nature, implying that they might be available for varying intervals of time, with frequent changes of their availability status. Also, the components tend to be heterogeneous and are distributed in nature. Hence, a software component framework [1] for Grid must fulfil the following

* The material presented in this paper is based upon work supported by the U.S. Office of Naval Research under award number N00014-01-1-0746.

requirements; a) Should be able to tackle heterogeneity in language, model, technology and architecture, etc., b) Allow a way of dynamic discovery of components, c) Provide a means for ascertaining non-functional attributes such as QoS of individual components as well as the integrated system, and d) Provide a user friendly mechanism for the system integration. In the current Grid scenario, a user developing a Grid application from pre-built components has to either write scripts in a XML representation [6], write scripts in a domain specific language [7, 9], or employ application workflow diagrams and graphical modelers [3, 8].

Conforming to the first approach, ICENI [6] provides a component based framework for creating Grid applications from pre-built components, discovered from private as well as public meta-repositories. Whenever a new component is developed, a component specification is created in terms of a CXML (Component – eXtensible Markup Language) [10] document, describing the component’s behaviour and interface. Implementations of the specification are placed in meta-repositories, with meta-data describing their performance characteristics and resource requirements. Based on a problem definition, composition of these implementations to form a Grid application is described in terms of an application description document, which is a CXML specification of the complete component composition. At runtime, the application description document is converted into an active Java representation by utilizing the component specification meta-data within the repository. The run-time representation is used to map the application requirements into available resources, based on requirements’ and implementations’ meta-data. While this approach does attempt to provide a component based Grid framework, for satisfying a few of the aforementioned requirements for a Grid framework, it does not succeed on several fronts. Firstly, it does not tackle heterogeneity at the component model level, only at the language level. The component CXML document does not provide a comprehensive enough QoS catalogue for comparing and matching components attributes, or for prediction at the component and system level. Since CXML does not accord the flexibility to express an application in terms of a hierarchy of possible subsystems, even a small change in the problem definition implies that the application CXML has to be rewritten.

One script based approach incorporated in GRADS [9] aims at providing domain specific high-level programming systems for problem solving environments, by which end users can rapidly develop new applications using standard notations of their problem domains. Here, the pre-built components are organized into optimized libraries, using a set of library design and specification strategies. Also, the application library is annotated with the following details; a) program transformation specifications detailing how program sequences can be replaced with equivalent, but more efficient sequences and b) sample calling programs illustrating typical usage patterns. In a separate step, mappings from scripting

languages to library implementation language are provided. The scripting languages enable the usage of components as primitive objects and define operations on them. A translator generator processes the enhanced library for hours or days and produces an executable. Using any of the allowed scripting languages, the user has to write an application script involving operations, initiation and configurations of the primitive objects to construct a Grid application. The scripts are then translated and compiled using the above domain specific translator. While the approach promises significant improvement in performance issues, it does not provide a software component framework (as in [1]) per se. As a result, there are no facilities for discovery of components, prediction of QoS of individual components and integrated system. Even though end users can develop new applications using their domain specific notations, there is a significant amount of latency curve associated with learning a new scripting language. The developed applications are invariably individualistic handcrafted solutions.

XCAT3 [7] is another script based framework that emphasizes distributed computing and provides Grid and Web Services connectivity to CCA (Common Component Architecture) [11] based on the OGSA. It provides a component based framework by which components (CCA components and/or Grid services based on OGSA) can be instantiated and connected together. Each component contains *provides-ports* indicating the functionality the component provides to other components and *uses-ports* indicating the needed functionality from other components that the component needs to function. As a result, each component in the XCAT3 framework consists of port interfaces, port implementations and an implementation. *Builder* services APIs in Java are provided, by which instances of components can be created and composed together to form a distributed application. Also, APIs for querying services of components, destroying component instances, and invoking methods on instantiated components are provided. While this approach is promising, the resulting applications are again handcrafted solutions. Further details of XCAT3 and OGSA in particular, are presented in the Section IV.

In an application workflow approach, for example in CrossGrid [8], the user supplies an initial application workflow document, detailing the components, their interactions and the workflow. Here, components are CCA-based, are developed independently, and are registered with OGSA registries. A flow composer parses the user workflow diagram, performs component lookups based on Port type or ID attributes and builds different final workflow documents with every distinct set of matched components. Finally, the user can choose a final workflow document corresponding to his view of the integrated system. This approach has intrinsic limitations similar to the script based approaches in that it places undue importance on the expertise of the user, does not offer any QoS testing and finally results in handcrafted,

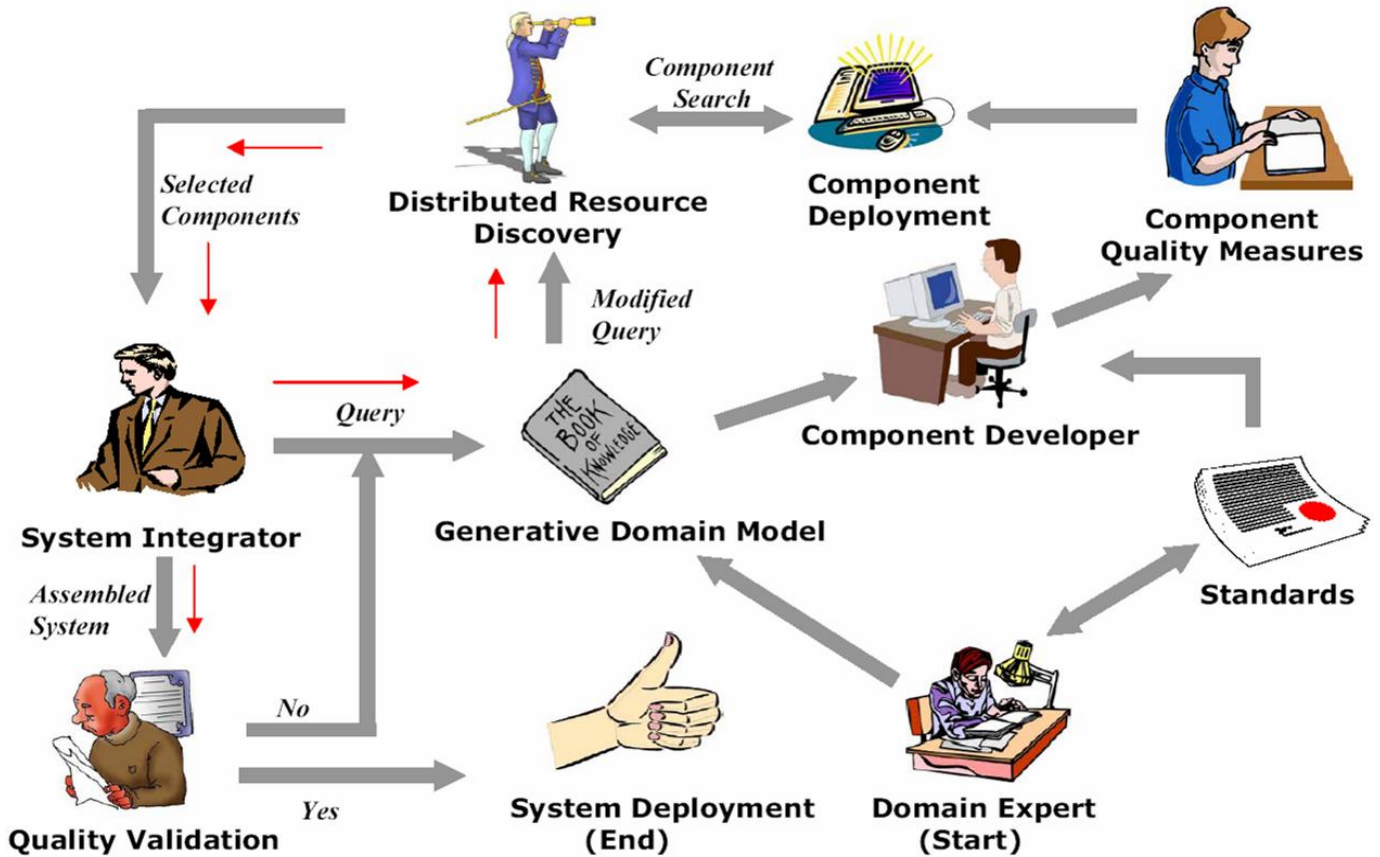


Fig. 1 GridFrame Process

individualistic solutions with restricted reusability. In Triana [3], graphical modelers and toolkits are employed, which provide the user a higher level of abstraction than application workflows. A user creates an application by dragging, dropping and deleting components and associated relationships in a graphical window. Here, though a greater level of convenience than the application workflow approach is accorded to the user, the other workflow limitations still remain. Other approaches like ECSF [12] provide a distributed computing paradigm suitable for multidisciplinary Grid applications, but are also limited by the same problems since their underlying principles are not based on the component paradigm. GridLab [13] does not provide a component creation framework, but focuses on providing high level application toolkits that interface between user applications and Grid middleware packages like Globus [2].

To summarize the related approaches, many challenges such as heterogeneity of components, their resource discovery and QoS prediction etc., associated with creating a component based Grid framework have currently not been addressed satisfactorily. If Grid has to become omni-present, in both scientific and commercial domains, these challenges need to be effectively addressed. One possible approach to addressing these challenges is by creating a comprehensive framework, incorporating solutions to the indicated problems.

III. GRIDFRAME APPROACH

As a part of a related effort called UniFrame [14], the principles behind addressing some of the aforementioned challenges have been developed. UniFrame is a component-based framework for interoperation of heterogeneous distributed components. In this paper, a symbiosis of the principles of UniFrame and Grid to form a component based Grid framework known as *GridFrame* is proposed. The key research issue that GridFrame addresses is the conception of a semi-automated Component Based Grid System (CBGS) development process involving the dynamic discovery of distributed Grid components, generation of the composed system and validation of quality requirements

GridFrame differs significantly from current Grid approaches by relying on an expert created generative domain model (GDM) [15]. Experts from the particular domain create the GDM containing the details of the distributed Grid application under consideration. The GDM contains details of the software architecture of families of possible systems in terms of the constituent software components, descriptions of the component characteristics and interactions, rules for the prediction and monitoring of quality of the constituent components as well as the integrated system. A reliance on a GDM has many advantages; a) it is created by domain experts, thus, end users are abstracted from domain knowledge expertise and required skills, b) model for

component developers to create individual components, and c) it provides rules for the composition and decomposition of components with associated quality of service. A component developer for an application consults the GDM and creates component implementations using the listed specifications. It is anticipated that many such components for a particular application with possibly different QoS, will be developed and deployed over a network.

A domain expert creating a GDM for a Grid application has to follow the GDM development process as outlined in [16]. The GDM development process consists of three phases; i) domain analysis - establishing domain scope, identification of functional and QoS requirements and mapping of relevant domain concepts, ii) domain design - development of common layered architecture for a family of possible systems and QoS related models, and iii) ordering design - design of ordering schemes for ordering a component-based system from the family of possible systems. Using this process, a GDM for the domain is developed. The GDM consists of three parts: general information, which includes a description for the modelled domain; a problem space, used by an application programmer to specify the needs; and a solution space, which contains various models including configuration knowledge to provide solutions for a CBGS family. Further details of the GDM are given in the case study section.

Component specifications are described by an associated Unified Meta-component Model (UMM) [17]. UMM has three parts: a) components, b) service and its guarantees, and c) infrastructure. A component in UMM is considered to be a tuple consisting of: a) inherent attributes - bookkeeping information such as name, description, etc., b) functional attributes - interface, pre-post conditions, algorithms, etc., c) non-functional attributes - supported QoS parameters and values with corresponding deployment environments, d) cooperative attributes - details of the collaborations of systems in which the component participates, e) auxiliary attributes - special features such as mobility, security, fault-tolerance, etc., and f) deployment attributes - configuration, initialization information.

The second part of the UMM is the service and associated guarantee of delivering that service. While realizing a CBGS from a set of independently created components, it is necessary to reason about the quality of the integrated CBGS. The quality of the integrated system translates into the quality of service offered by each component and of their interactions. Hence, it is necessary that a component provide a pre-determined level of quality of both its functional and non-functional features. For doing so, the UMM requires a component developer to specify the QoS parameters that are applicable to a particular component and the ranges that the component can guarantee when operating under a certain execution environment [17].

The third part of the UMM is the infrastructure that supports the creation, publication, deployment, and location of the components and their services. This infrastructure is provided by Grid Resource Discovery System (GRDS) based

on [18], which is the infrastructure that supports the creation, publication, deployment, and location of the components and their services. The discovery process in GRDS is scoped administratively implying that it locates services within an administratively defined logical domain. A domain is defined as industry specific markets such as Information Filtering Services, Health Care Services, and Financial Services, etc. The GRDS architecture consists of the following entities: a) head-hunters for discovering component specifications, b) containers for component registration, and c) components. Components are implemented in accordance with component models such as Microsoft .NET, Java RMI, CORBA, etc., and are registered with the binding service of that model. The binding services are modified Grid containers such as Globus J2EE container, .NET container etc. Headhunters periodically communicate with these Grid containers and retrieve and store specifications (service data) of registered components into their local meta repositories. For more details about UMM components, service and infrastructure, please refer to [14], [19] and [18] respectively.

Components offer services, indicate and guarantee the quality of their services, and hence, it is necessary to facilitate the publication, selection, measurement and validation of the component and system QoS values. The Grid Quality of Service Framework (GQoS) based on [17] provides the necessary guidelines for the component developers and system developer using GridFrame. The GQoS is made up of three parts: a) QoS catalogue - collection of possible QoS parameters such as end-to-end delay, throughput etc., b) specification and measurement of QoS and c) composition/decomposition models for QoS parameters. For further details, please refer to [14] and [18].

Fig. 1 gives an illustration of the GridFrame process. In the beginning of the process, a Grid system developer, developing a CBGS, for a specific application issues a query containing the requirements for the CBGS. The query can comprise of functional requirements as well as non-functional QoS requirements such as end-to-end delay, throughput, etc. The query processing consults the GDM for the design of an appropriate CBGS and may divide the query into many sub-queries, each corresponding to a single component UMM specification. These sub-queries are passed to the GRDS which searches for appropriate matching components. If components are found, they are displayed to the system developer. The system developer decides on the components to be included in building the system, based on various criteria such as offered QoS. Also, each component provides an associated testing mechanism, which can be used to dynamically test the QoS characteristics of the component. These dynamic test values can be judged against the component developer's specifications of the component. After the system developer selects his choice of components, the generation of the integrated system is carried out by the GridFrame System Integrator [16] using the selected components.

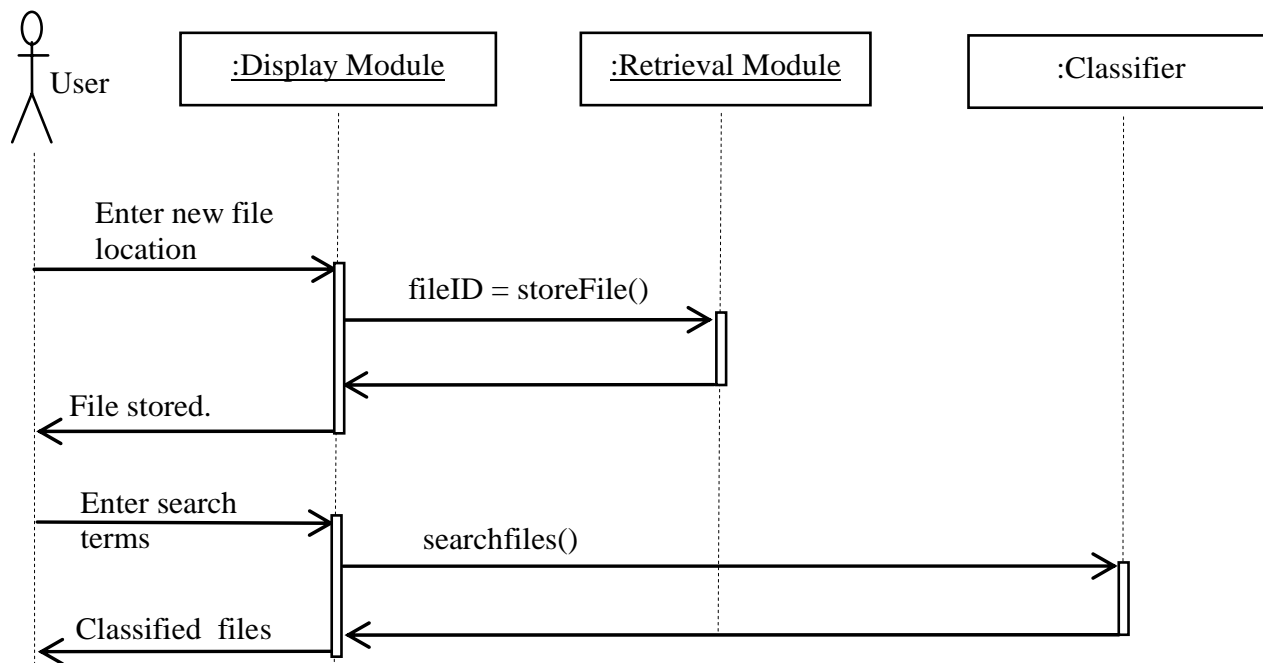


Fig. 3 Partial Sequence Diagram for Search

the previously identified Grid services that would aggregate to form a DIFS Grid service. Fig. 2 illustrates an example of the process by which a user can build a DIFS system using Grid services. It contains the following steps:

2) The user contacts a known registry to identify service providers who can provide the required services and presents a list of requirements including cost and performance.

3) The handles for needed service factories that match user requirements are returned to the user.

4) The user supplies instantiation details such as needed operations, etc., and initial lifetimes for the service instances.

5) If agreeable, the service providers create service instances with user supplied details.

6) Using the service handles, the user writes application programs for aggregating the services to form a DIFS system. Visions of enterprises using Grid Services approach to dynamically compose new applications such as above to address the specific needs of the business at any point in time have been painted. But there are several limitations with this approach, particularly in regard enterprise applications. The resulting new applications are basically handcrafted solutions with limited reuse. Any slight change in the problem definition, for instance using a .NET display component, if a previously used Java component is not available, will entail a complete rewrite of the previous solution. Also, users do not have options for any preliminary testing of the integrated application, implying that they cannot make any informed decisions about the QoS of the integrated application before the actual deployment. In addition, most of the techniques for discovery of components assume that the components are homogeneous in nature and rely on simple interface matching and component context dependencies, which are not sufficient

enough for a process, which is a precursor for composing high confidence Grid systems. Also, the approach assumes a high level of programming skill of users, which is typically not the case with mainstream Grid users. These are serious drawbacks, particularly considering that mainstream domains such as enterprise applications have stringent requirements about quality and reusability of applications.

B. Creating a DIFS system with GridFrame

For the sake of brevity, the focus is mainly on the overall outline of carrying out the development of a partial GDM as well as the discovery mechanisms of GridFrame possibly resulting in the omission of in depth details, which can be referred to, using the associated references.

1) *GDM process*: Due to space constraints, only some of the aspects of GDM like feature diagrams and use-cases depicting the configuration knowledge of the integrated system are shown here. In the GDM, the components making up the DIFS system are identified along with their functional characteristics such as required interfaces, provided interfaces, etc., and non-functional characteristics such as QoS metrics. In addition to the feature diagram, the GDM contains sequence diagrams, which capture the behavioural aspects of the system. Sequence diagrams, such as Fig. 3 illustrate the interaction of components in the system with each other as well as with users. Fig. 5 shows a feature diagram illustrating the DIFS family of sub-systems which can be possibly built with the identified components.

The details about the concept of features and the notation used for describing a feature diagram are proposed in [16]. The given feature diagram indicates possible architectural alternatives for a DIFS. For example, two possible alternatives for a DIFS could be: version (a) made up of RM, TM, RP, CL,

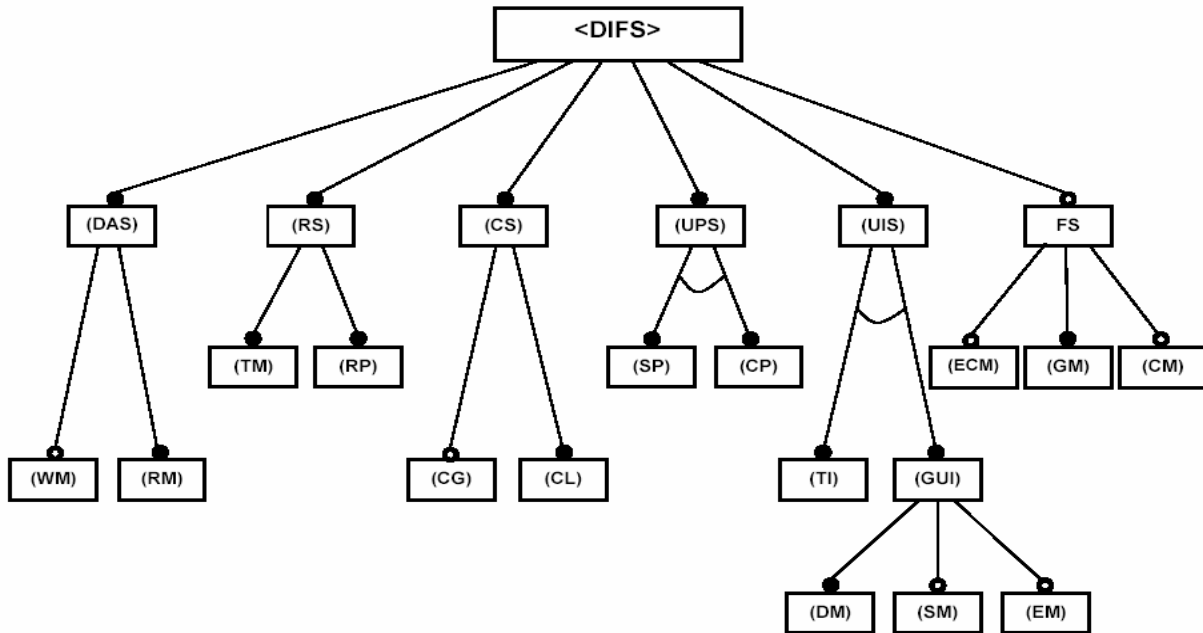
SP, TI and version (b) made up of WM, RM, TM, RP, CG, CL, CP, DM, SM, EM, GM, ECM, CM. As indicated earlier, depending upon the input query presented by the system integrator, an appropriate alternative will be selected during the system development process. Each node in the feature diagram indicates an abstract component, which will be described by its corresponding UMM specification. The component specification in UMM is a multi-level contract [19] with bookkeeping information such as component id, domain name, and algorithmic, technological information such as function name, algorithm name etc. For example, a partial UMM-specification for a typical classifier could be:

1. Component Name: Classifier
2. Domain Name: Information Filtration
3. System Name: InformationFilter
4. Informal Description: Provide classification service for documents.
5. Computational Attributes:
 - 5.1 Inherent Attributes:
 - 5.1.1 id: N/A 5.1.2 Version: version 1.0 5.1.3 Author: N/A
 - 5.1.4 Date: N/A 5.1.5 Validity: N/A 5.1.6 Atomicity: Yes
 - 5.1.7 Registration: N/A 5.1.8 Model: N/A
 - 5.2 Functional Attributes:
 - 5.2.1 Function description: Act as classification server for documents in system.
 - 5.2.2 Algorithm: N/A 5.2.3 Complexity: N/A

- 5.2.4 Syntactic Contract
 - 5.2.4.1 Provided Interface: IClassification
 - 5.2.4.2 Required Interface: NONE
- 5.2.5 Technology: N/A
- 5.2.6 Expected Resources: N/A
- 5.2.7 Design Patterns: NONE
- 5.2.8 Known Usage: Classification of documents
- 5.2.9 Alias: NONE
6. Cooperation Attributes:
 - 6.1 Preprocessing Collaborators: Representor
 - 6.2 Postprocessing Collaborators: NONE
7. Auxiliary Attributes:
 - 7.1 Mobility: No 7.2 Security: L0 7.3 Fault tolerance: L0
8. Quality of Service Attributes
 - 8.1 QoS Metrics: throughput, end-to-end delay
 - 8.2 QoS Level: N/A 8.3 Cost: N/A 8.4 Quality Level: N/A
 - 8.5 Effect of Environment: N/A 8.6 Effect of Usage Pattern: N/A
9. Deployment Attributes: N/A

Fig. 4 A UMM example for Classifier

Once the GDM has been developed, component developers are free to develop and deploy components using their choices of technology, language, etc., according to the specifications in the GDM. The developed concrete components have to strictly adhere to the GDM abstract specifications, but can be



Legend:

DAS – Data Acquisition Service
 WM – Wrapper Module
 RM – Retrieval Module
 RS – Representor Service
 TM – Thesaurus Module
 CS – Classifying Service
 CG – Centroid Generator

SP – Simple Profiler
 CP – Complex Profiler
 UIS – User Interface Service
 TI – Text Interface
 GUI – Graphical Interface
 DM – Display Module
 SM – Statistical Module

CL – Classifier
 UPS – User Profile System
 GM – Group Manager
 EM – Editing Module
 FS – Federation System
 ECM – Economic Module
 CM – Common Module

Fig. 5 Feature Diagram of DIFS

implemented in different technologies, algorithms etc. with corresponding QoS values. For example, one Representor Module (RM) can be implemented in .NET technology using a vector space model [20] with 340 ms turnaround time while another RM can be implemented in Java RMI using a different model with 320 ms turnaround time, with corresponding QoS attributes

2) *Discovery of components and Integration of the system:* After the creation of the GDM and deployment of components, a system developer can query for an instance of a system using a tabular graphical interface [16], containing different options for the different possible systems. For example, the options could be a basic DIFS with minimal functionality incorporating instances of RM, TM, RP, CL, SP, TI and GM or an advanced DIFS with increased functionality incorporating instances of WM, RM, TM, RP, CG, CL, CP, DM, SM, ECM, GM. For example, the system developer might query for a simple DIFS with QoS values such as the maximum permissible end-to-end delay and minimum throughput for the system specified as 1800 ms and 400 op/s respectively. Using the decomposition model in [22], the given QoS requirements for the whole system are decomposed into the QoS requirements for each of the constituent components. By means of the GDM and the QoS requirements, for each of the components making up the chosen system, a query is created. These queries are presented to GRDS for discovering concrete instances of the components, which can match the requirements.

When the GRDS receives the requests, a subset of headhunters in the specified domain (in this case, distributed information filtering) is contacted for concrete instances of the components. These headhunters search their local meta-repositories and perform syntax, semantic and QoS matching of the stored specifications with the queries. Each query has an associated timestamp, depending on which the queries can be propagated to other headhunters. For details about selection, propagation and matching algorithms of headhunters, see [18]. As explained in the GridFrame process, the system developer chooses from among the listed components on basis of QoS values, (available from the service data), and uses the GridFrame System Integrator to test and build the integrated system.

A brief comparison of the two approaches suggests the following:

1) As opposed to handcrafting, the use of a GDM in GridFrame enables the creation of standardized solutions by which the reusability of individual components as well as the integrated system is improved.

2) Quality of service theme is maintained throughout the GridFrame process, as a result of which predicting and monitoring of component performance at the component level as well as system level is possible.

3) GridFrame accommodates heterogeneity by which components can be implemented in different models and technologies.

4) By providing a semi-automated framework for composing services, GridFrame ensures that user intervention is minimized, enabling novice end users to integrate systems.

5. CONCLUSION

The proposed framework provides a semi automated approach for building Grid systems from pre-built Grid services using concepts of software engineering. Using the framework, it is possible for end users to both predict and reason about the quality of the integrated system as well as the individual services. Although a simple example is provided here, the principles are general enough to be applicable for both mainstream and research Grid projects. The development of Grid systems involves both construction and deployment of the system. Here, only the construction issues of a component based Grid system using a GDM were discussed. Utilizing the GDM for deployment issues such as assessing hardware resource requirements, selecting ideal resources, etc., is the focus for current research. The GridFrame process has been investigated using the Globus toolkit and the current UniFrame infrastructure by means of trivial examples. While the results are promising and show that such a process is plausible, validation on a realistic, large scale scientific or mainstream Grid application is one of the key research goals for current and future efforts.

6. REFERENCES

- [1] C. Szyperski, D. Gruntz., S. Murer, *Component Software - Beyond Object-Oriented Programming*, Second Edition. Boston: Addison-Wesley/ACM Press, 2002.
- [2] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure WG, *Global Grid Forum*, June 22, 2002.
- [3] I. Taylor, M. Shields, I. Wang, and R. Philp, "Distributed P2P computing within triana: A galaxy visualization test case," *IPDPS*, 2003.
- [4] G. Allen, W. Bengler, T. Goodale, H. Hege, G. Lanfermann et al, "The Cactus Code – A Problem Solving Environment for the Grid," *Proceedings of the 9th IEEE Int'l. Symposium on High Performance Distributed Computing*, pp.253-263, Pittsburgh, 2000.
- [5] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid", *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003.
- [6] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field and J. Darlington, "An Integrated Grid Environment for Component Applications," *Second International Workshop on Grid Computing 2001*, pp. 26-37, November 2001.
- [7] S. Krishnan, and D. Gannon, "XCAT3: A Framework for CCA Components as OGSA Services," *Proceedings of HIPS 2004*, pp. 90-97, April 2004.
- [8] M. Bubak, K. Gorka, T. Gubala, M. Malawski, K. Zajac, "Automatic Flow Building for Component Grid Applications," *Fifth International Conference on Parallel Processing and Applied Mathematics*, 2003.
- [9] K. Kennedy, B. Broom, K. Cooper, J. Dongarra, R. Fowler, D. Gannon et al, "Telescoping Languages: A Strategy for Automatic Generation of Scientific Problem-Solving Systems from Annotated Libraries," *JPDC*, Vol. 61, No. 12, pp. 1803-1826, Dec 1, 2002.
- [10] N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field and J. Darlington, "Optimisation of Component-based Applications within a Grid Environment," *Parallel Computing*, n.12, p.1753-1772, December 2002.
- [11] D. Gannon, S. Krishnan, L. Fang, G. Kandaswamy, Y. Simmhan, and A. Slominski, "On Building Parallel and Grid Applications: Component Technology and Distributed Services," *CLADE 2004*, June 2004. URL : <http://www.extreme.indiana.edu/xcat/publications/clade04-cca-grid.pdf>.

- [12] P. Bangalore, "An Open Framework For Developing Distributed Computing Environments For Multidisciplinary Computational Simulations," *PhD thesis*, Mississippi State University, May 2003.
- [13] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser et al, "The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid," Unpublished. URL: <http://www.gridlab.org/WorkPackages/wp-1/publications.html>.
- [14] R. Raje, M. Auguston, B. Bryant, A. Olson, C. Burt, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components," *Proceedings of the 2001 Monterey Workshop*, pp. 109-119, Monterey, California, 2001.
- [15] K. Czarnecki. and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
- [16] Z. Huang, The UniFrame System-level Generative Programming Framework., *MS thesis*, IUPUI, CIS Department, 2003.
- [17] G Brahmamath, R. Raje, A. Olson, B. Bryant, M. Auguston, C. Burt, "A Quality of Service Catalog for Software Components." *Proceedings of the Southeastern Software Engineering Conference*, Huntsville, Alabama, 2002.
- [18] N. Siram, "An Architecture for the UniFrame Resource Discovery Service," *MS thesis*, IUPUI, CIS Department, 2002.
- [19] A. Beugnard., J. Jezequel, N. Plouzeau. and D. Watkins, Making Components Contract Aware. *IEEE Computer*, 32(7):38-45, July 1999.
- [20] R. Raje, M. Qiao, S. Mukhopadhyay, M. Palakal, J. Mostafa, "SIFTER-II: A Heterogeneous Agent Society for Information Filtering," *Proceedings of ACM Symposium on Applied Computing, SAC'01*, pp: 121-123, Las Vegas, Nevada, 2001.
- [21] C. Sun, "QoS Composition and Decomposition Model in UniFrame," *MS thesis*, IUPUI, CIS Department, 2003.