A Flexible Event-Driven Architecture for Peer-to-Peer Based Applications.

Conference Paper · January 2011

Source: DBLP

CITATIONS
0

READS
530

5 authors, including:

Edward de Oliveira Ribeiro

21 PUBLICATIONS **53** CITATIONS

SEE PROFILE

Genaina Rodrigues

University of Brasília

67 PUBLICATIONS **695** CITATIONS

SEE PROFILE

Célia Ghedini Ralha

University of Brasília

148 PUBLICATIONS **660** CITATIONS

SEE PROFILE

A Flexible Event-Driven Architecture for Peer-to-Peer Based Applications

Leone Parise Vieira da Silva, Rajiv Geeverghese, Edward de Oliveira Ribeiro,
Genaína Nunes Rodrigues and Célia Ghedini Ralha
Computer Science Department, University of Brasília
POBox 4466, Brasília/DF, ZIP 70.904-970, Brazil
{leone.parise,rajiv.unb,edward.ribeiro}@gmail.com, {genaina,ghedini}@cic.unb.br,

Abstract

Over the last decade, we have seen an increasing interest in the event-driven architecture (EDA) approach. EDA allows the transmission of events among loosely coupled and highly-distributed software components, which is totally adequate to peer-to-peer based applications mainly considering features of scalability and flexibility. In this paper, we present the initial steps towards the design and implementation of a peer-to-peer based EDA, which is intended to reduce the coupling between application layers, and achieve a better distribution of responsibilities between the architecture elements and the environment. Our flexible EDA framework, proposed in this work, inherits important features such as scalability, decentralization and fault-tolerance from the peer-to-peer domain. For a preliminary evaluation of our approach, two experiments using the EDA framework were conducted in different domain applications.

1. Introduction

Event-driven architecture (EDA) is a style of software architecture based on real time flows promoting the production, detection, consumption of, and reaction to events [8]. This architectural pattern may be applied by the design and implementation of applications and systems, which transmit events among loosely coupled and highly-distributed software components and services.

In EDA, events processing may follow three different styles: simple, stream and complex [8]. In simple processing, when a meaningful event happens it initiates downstream actions. Stream processing deals with multiple streams of event data aiming at identifying the meaningful events within those streams. Complex event processing evaluates a confluence of events and then takes action. The events, which may be meaningful or ordinary, may cross event types and occur over a long period of time.

Peer-to-peer (P2P) networks offer an appropriate paradigm for developing efficient distributed systems and applications. In a P2P network, all processes involved in a work or activity have similar roles, interacting cooperatively as peers with no distinction between client and server computers or the infrastructure where they perform [13]. P2P applications are composed by a great number of processes executing on a distributed peer way. The communication pattern among processes is a key point which depends on the application necessities.

Important aspects of P2P applications must be fulfilled through software development frameworks encompassing loose coupling, scalability, decentralization and fault-tolerance skills. The convergence of EDA and P2P is a natural outcome of the recent evolution of distributed systems, since many of the challenging standards issues are closely related. This convergence creates best practices that enable interoperability between networking systems for P2P applications. Therefore, a flexible EDA is essential to improve communication, allowing the exchange of objects between processes in a natural way.

In this work, we propose a flexible EDA for peer-to-peer based applications, keeping the principles of modularity, loose-coupling, well-distribution and P2P technology independence. We implemented a basic version of the proposed EDA, which accommodates the simple event processing style. We evaluate our proposal using two different domain applications, so as to explore distribution through parallelization of the applications tasks. Our results show the easy integration with P2P technologies, while seamlessly preserving EDA principles.

The rest of the paper is organized as follows. The proposed EDA is presented in Section 2, focusing on the architecture (Section 2.1), with the implementation aspects (Section 2.2) and the developed interfaces (Section 2.2.1). The description and the results of the experiments carried out are described in Section 3. Related work is presented in Section 4. Finally, Section 5 concludes the paper and presents our future research directions as well.

2. Our Proposed EDA

The main principle adopted during the definition of the proposed EDA is to divide the development of applications in different layers, providing communication among them through the use of an event bus module and their respective interfaces. This principle allows flexibility, loosely coupling between application layers, better distribution of responsibilities between the architecture elements and the environment, maintaining functionality even when using mobile devices through a TCP network and personal computers in a P2P approach. Together with the P2P applications, other important skills are maintained e.g. scalability, decentralization and fault-tolerance.

The rest of this section presents the proposed EDA architecture, including implementation aspects with the defined interfaces and P2P technologies.

2.1. The Architecture

Our proposed EDA presents four layers: Application, Environment, Network Adapter and Event Bus, as presented in Figure 1. The Application layer is the upper platform layer, which allows the development of different P2P applications, using our EDA framework as a middleware of development.

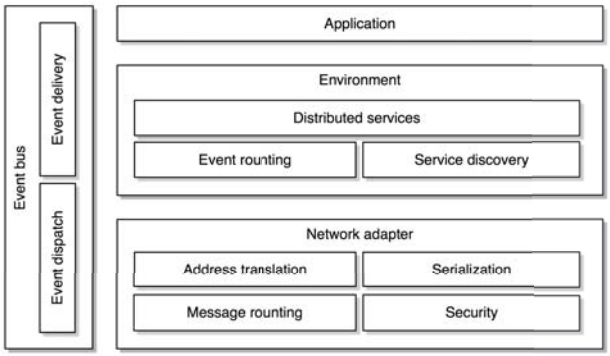

Figure 1. The four-layered architecture.

The Environment layer is composed by the Distributed Services, the Event Routing and the Service Discovery modules. The Environment layer abstracts the modules, allowing the communication and localization of services to be done in a transparent way. In other words, the application issues events in the Environment, which is responsible to notify listeners, without direct communication and connection among different services. In this way, the applications do not know if they are in the same or in different machines. This abstraction feature would help the development of different applications, including software agents, allowing their interaction without the communication bottleneck and independent of the network topology being used.

Apart from offering the described abstraction feature, the Environment layer has an Event Routing and a Service Discovery modules to publish and locate services in a distributed way. Services like distributed database, agent transport, integration of mobile devices can be developed and coupled to the environment transforming it in a more integrated framework. Lots of applications using for example multiagent approach with heavy data, information and knowledge requirements such as bioinformatics, forensics, automated reasoners on the Web may benefit from the features above mentioned.

The Network Adapter is responsible for integrating the Environment to the underlying network, acting as an interface between the P2P middleware and the other EDA framework layers. This layer is composed by the Address Translation, Serialization, Message Routing and Security modules. The integration among the chosen P2P middleware and our EDA framework is done by assigning a unique identifier of each addressable component of the platform, which is represented by the UUID (Universally Unique IDentifier) object of Java API. The P2P middleware implementation uses a table in order to translate these identifiers into other address type, e.g. IP address or P2P network node. In this way, another P2P implementation, such as Chord [17] or Tapestry [10], can be couped to our proposed EDA without modifying other modules.

We should note that, if the chosen P2P middleware has the necessary features of address translation, message routing, serialization and security, then it is not necessary to implement the Network Adapter layer. But, in case it is necessary, it receives the events from the upper platform layers, serializes as a message in the network and sends to the application. Whenever the adapter receives a message, it unserializes as an event, which is sent to the event bus module so that the upper platform layers receive the communication.

Finally, the Event Bus layer is responsible for the Event Dispatch and Event Delivery modules. An event flow starts with the event being generated and culminates with the execution of any downstream activity. The processing of this flow can be described in logical layers. The four logical layers are: event generators, event channel, event processing, downstream event-driven activity [8]. The event processing flow used in our work includes the event generator, event channel, event processing and downstream event-driven activities. The flexibility at each logical layer is the power of event, since the way events are created, how they are transmitted, received and processed is transparent to the P2P applications developed over our proposed EDA.

2.2. Implementation Aspects

The methodology used to define the EDA was based in design patterns with the objective to support its implementation. In order to keep the principles of modularity and loose coupling of event-driven architecture we applied six design patterns, which were the most suitable to the principles of the proposed EDA: $AbstractFactory$, $Singleton$, $Observer$, $Decorator$, $Command$ and $Mediator$ [5]. To validate the proposed EDA, we implemented a prototype in Java at the Computer Science Department of University of Brasília. The prototype has 67 classes and 3,600 lines of documented code.

All the communication in the proposed EDA is done by events, which reduces the object coupling, allowing flexibility and scalability to the architecture. In this sense, the communication has two important elements: the event and the listener, which always come in pairs, since the event carries the object message content and the listener defines which object has to listen specific event.

The Event Bus layer is responsible for the events dispatch and delivery in the platform, which is implemented through three design patterns: $Singleton$, $Observer$ and $Mediator$. $Singleton$ guarantees only one Event Bus instance, while $Observer$ records all listeners and notify them of sent events. The Event Bus layer is also the object mediator in the communication of the platform according to the $Mediator$ pattern. The Event Bus interface is described in Section 2.2.1.

The possibility to execute commands between objects through the Event Bus layer allows the method call between different distributed components in the P2P network. Using $Command$ design pattern we have implemented two special events: $CommandEvent$ and $ResponseEvent$, which inherit from the $Event$ abstract class. Though these events work just like the others, their expected behavior is different. All object that receives a $CommandEvent$ may execute the command and send the results by the $ResponseEvent$. All objects that send a command may wait for an answer during a limited time, and in case the answer does not come, a timeout event is send to the object to notify the expiration time. A method $getId()$ is used in the $Event$ class to identify events.

2.2.1 Interfaces

In order to allow the flexibility and the loosely coupled features of our architecture, considering the application, environment, network and event processing, along with the EDA layers intercommunication, we have developed some interfaces. We present the major interfaces implemented: (i) $Platform$ – interface where all layers and modules are integrated; (ii) $EventManager$ – the event interface that handles the event processing; (iii) $Environment$ – the environment interface to deal with services discovery and distribution; (iv) $NetworkAdapter$ – network interface to deal with the P2P processes.

The $Platform$ interface is the core of our EDA framework. It defines the management of the platform configurations and disaggregates all components (layers and modules) into implementation interfaces, allowing uncoupling and flexibility. This interface follows the $AbstractFactory$ design pattern. Therefore, every component can be reimplemented and replaced without affecting the operation of the other components. The $Platform$ component is also a $Singleton$, which helps and unifies the access of objects among all $Platform$ components, helping the unitary and integration tests.

The $EventManager$ is one of the most important developed interfaces that implements the Event Bus layer, presented in Figure 2. Note that the $EventManager$ interface has two methods: $addlListener()$ – responsible to register listeners and $fireEvent()$ – responsible to communicate an event to a group of listeners. The $EventManager$ interface keeps the reference of all the registered listeners. Whenever the event bus receives an event, it extracts all the objects that inherit the class obtained from the method $getAssociatedListener()$ and notify them in a separate thread. The $EventManager$ implements the $Mediator$ design pattern. Suppose ObjectA wants to communicate to ObjectB, then ObjectA creates a $ConcreteEvent(string)$ and send it to $EventManager$ using the $fireEvent()$ method. The $EventManager$ that has ObjectB registered in the listeners list, communicates the event to ObjectB using $onEvent()$ method.

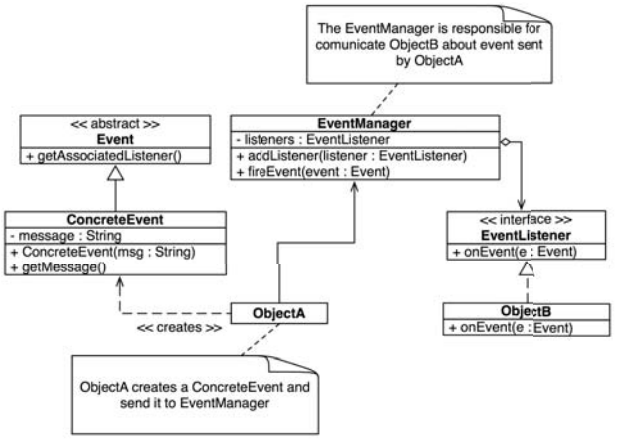

Figure 2. Implementation of the $EventManager$ interface.

The $Environment$ interface defines the environ-

ment in the EDA framework, which has two methods: $requestService()$ and $destroyService()$. The $requestService()$ is responsible to publish and get an instance of the published service.

$NetworkAdapter$ interface deals with the P2P processes. The $NetworkAdapter$ contains two send methods, one to send simple events and the second to send commands and the respective temporary listener to the network. Related to the P2P approach some middlewares were studied to integrate our framework. Important aspects of scalability, fault-tolerance and self-organization were considered, including Chord [17], Can [9], Tapestry [10] and Pastry [12].

We could have used any of the above mentioned P2P middlewares. However, we chose Pastry together with Scribe [2] considering their comprehensiveness. Pastry uses routing mechanisms to achieve great scalability, while Scribe depends on Pastry to route messages to the destinations. Scribe [2] is a typical pub-sub system built on top of Pastry, which leverages its scalability, routing efficiency and self-organization capabilities [15]. The event processing, including dispatch, delivery and routing of events, and the service discovery in our EDA is executed over the Pastry and Scribe mechanisms, making use of the benefits provided by such combination. In this sense, the $NetworkAdapter$ interface is responsible for encapsulating an event through the $PastryEvent$ and send to Pastry.

3. Experiments and Evaluation

During this research project two experiments using the EDA framework prototype were conducted: the turtle and the bioinformatics projects. The objective to use two experiments is to evaluate our framework.

3.1. The Turtle Project

The turtle project is the simple sum of harmonic series, which apparently converge to a number, but is proved mathematically that it varies very slowly. Equation 1 presents the sum of first n elements of the series.

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \quad (1)$$

Table 1 presents the relationship between the lowest number of terms required for the sum of the series is equal to or greater than a specific number. Note that to reach a total value of 1,000 this amounts to 10^{434} terms which is a computational challenge to calculate the sum of the harmonic series term by term, in an iterative fashion.

Our solution was to build a distributed application using the "divide and conquer" and software agents approaches, using the facilities that the EDA framework offers. We

Table 1. Number of n terms sum.

k	least $n \mid H_n \geq k$
1	1
2	4
3	11
5	83
10	12,367
20	272,400,600
100	$\approx 1.51 * 10^{43}$
1,000	$\approx 1.10 * 10^{434}$

used blocks, with the start and size information, and distribute the blocks throughout the network to perform a partial calculation. To implement the distributed solution, two distinct roles become evident: (i) a central figure, to control the distribution of blocks in the network and the receipt of partial calculations; and (ii) a task force, specialized in the processing of the blocks. Each role is implemented by one particular agent, the $BossAgent$ and the $TraineeAgent$ (respectively) both agents have a parent in common, $AbstractAgent$, and inherit their behavior. The $AbstractAgent$ seeks the event services and discovery of services and adds to the platform using the $environment$ interface (Section 2.2.1).

The $BossAgent$ has an internal data structure to track which blocks are being calculated at the time and a variable that stores the next block, starting with the value 1. The block size is fixed, but can be changed before starting the agent. When the $BossAgent$ receives a block request from the $TraineeAgent$, it sends the next block to be processed and waits for the partial sum. Every 5 seconds, the $BossAgent$ uses a log tool to print the partial sum and the number of blocks processed in a file. Once created the agents, the application is started automatically via the exchange of events.

The computational platform used in this test bed (experiment) had eight computers with the same configuration: Pentium 4, 2.66 GHz, 1GB of RAM, and Ethernet 10/100 MB/s. The number of blocks processed with one computer is 209, with two computers is 418, with four computers is 827 and with eight computers is 1,612, considering the total execution time of 00:04:58. Note that the growth in the number of processed blocks is linear as presented in Figure 3.

3.2. The Bioinformatics Projects

P2P model is being used for applications that requires great amount of computer resources, since it offers some direct solution to modularity and scaling properties in large scale distributed systems. In this context, P2P approach is very important to bioinformatics. Thus, we have conducted

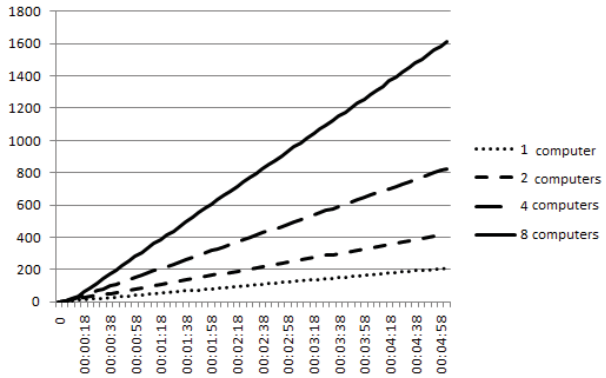

Figure 3. The number of blocks processed over time.

this experiment to illustrate the use of our EDA prototype in this area.

In this project we executed a particular bioinformatics application, BLAST, which is a family of algorithms for searching similarities between two biological sequences, widely used in genome projects [1]. We used real data of $Escherichiacoli$ or $E.coli$ bacteria, which is very known to bioinformatics community. The objective of the experiment is to apply BLAST algorithm to $E.coli$ sequences in order to validate EDA aspects of distribution, parallelism, and fault-tolerance. Although the $E.coli$ sequences is not expressive enough to justify high-throughput processing, our intention is to evaluate the abstraction of EDA proposal from the application perspective in different domains.

To implement the distributed solution, we used the same approach of the turtle project, presented in Section 3.1, according to the simple event processing style. The computational platform used in this test bed (experiment) had eight machines with the same configuration: Intel(R) Core(TM)2 Duo CPU E7500, 2.93GHz, 2GB of RAM and Ethernet 10/100 MB/s. Table 2 presents the execution time to process distributed BLAST over 6,400 $E.coli$ sequences. Note that the growth in time is linear, just like the turtle project presented in Figure 3.

3.3. Experimental Evaluation

Considering the fact that both experiments were carried out in different domains, extra effort was not necessary in the EDA layers. The modularity of our EDA framework was guaranteed by the defined layers and the modules. The provided interfaces (e.g., $Environment$, $EventManager$) assured the flexibility of our EDA framework, since it could encapsulate the application in both experiments, without generating extra implementation efforts to the application development. Other computer intensive P2P applications may use our EDA in a similar way.

Table 2. Time execution to process 6,400 sequences.

Number of Computers	Execution Time
1	00:24:44
2	00:12:25
4	00:06:16
8	00:03:07

In addition, our architecture promotes a high level of abstraction of the underlying P2P technology used, preserving flexibility. The EDA framework could benefit from the scalability and decentralization guaranteed by the P2P middleware. In this work, we have analyzed basic fault-tolerance features of our framework through the use of Scribe, which implements the services of the Network Adapter layers. The analysis consisted in repeatedly disabling the root node during our test beds. The *rendez-vous* point was reassumed without harming the processes computation.

As a preliminary evaluation of our approach the conducted test beds were satisfactory, but further experiments are necessary for a comprehensive evaluation, specially considering scalability and fault-tolerance features. The modularity and the extensibility of our framework may allow the use of other implementations for the security and reliability of the P2P processes at runtime, such as the work proposed by Spanoudakis [16] in the context of mobile P2P systems.

4. Related Work

Our proposed EDA functionally keeps the same principles presented in the EDA architecture, but the elements are organized in a different way, since our focus lies on P2P applications [8]. Pastry was chosen in this work due to its advantages over previous P2P frameworks as JXTA [19]. JXTA modular design and building blocks makes it relatively quick and simple to prototype P2P systems like document sharing and instant messaging in a timely fashion. Nevertheless, its design choices had severe impact on its scalability and reliability. JXTA is defined by a six layer XML based protocol stack that makes its implementations complex, heavyweight and slow [6].

Quite a few research work have focused on event-driven systems [3, 4, 7]. The EDA pattern has been usually associated to SOA, since they are complimentary, as SOA focuses on the decomposition of business functions and EDA focuses on business events [14]. Our approach to EDA is com-

prehensive since the development of P2P applications can be related to services through the decomposition of business functions.

Much research has been done to propose systems over structured P2P networks [18]. There are many applications developed using a P2P approach in various domains, particularly in bioinformatics [11]. However, as far as we are concerned, we could not find research work that integrates EDA approach to P2P networks with focus on a flexible architecture suitable for a multitude of various application domains.

5. Conclusion and Future Work

In conclusion, this work proposed a flexible EDA for P2P based applications. We implemented a basic version of the proposed EDA, which accommodates simple event processing style. We evaluate our proposal using two different domain applications exploring distribution of the P2P network. Our results an easy integration with P2P technologies, while exploring the features provided by the underlying P2P middleware used. Our preliminary results show that our EDA framework could be used seamlessly in two different domains.

As future work, we intend to expand the basic version of our current EDA implementation in order to deal with complex event processing style. Particularly, extend the implementation to deal with various and distinct event types simultaneously. For example, applications in web domain with events of different source types, video, text and image. We also plan to evaluate our framework for its flexibility using different P2P middlewares for validation purposes.

Another important future work is concerned to the extension of the current EDA core in order to transform the application module to encapsulate agents approach. In this direction, computer-intensive applications that may take advantage of multi-agent approach, through the use of rationality and distributed resources can take advantage of our EDA approach.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20:2002, 2002.

[3] S. Chakravarthy and R. Adaikkalavan. Events and streams: Harnessing and unleashing their synergy! In *Proc. of the 2nd DEBS, Italy*, pages 1–12. ACM, 2008.

[4] H. Dempo. Qos evaluations of distributed event orchestration system. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 22:1–22:5, New York, NY, USA, 2009. ACM.

[5] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[6] E. Halepovic and R. Deters. The costs of using jxta. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, P2P '03, pages 160–, Washington, DC, USA, 2003. IEEE Computer Society.

[7] A. Hinze, Y. Michel, and L. Eschner. Event-based communication for location-based service collaboration. In *Proc. of the 20th ADC - Volume 92*, pages 125–134, Australia, 2009. Australian Computer Society, Inc.

[8] B. M. Michelson. Event-driven architecture overview. Technical report, Patricia Seybold Group, OMG, 2006. http://www.omg.org/soa/UploadedDocs/EDA/bda2-2-06cc.pdf.

[9] S. Ratnasamy, P. Francis, S. Shenker, and M. Handley. A Scalable Content-Addressable Network. In *In Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[10] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-Free Global Data Storage. *IEEE Internet Computing*, 5(5):40–49, 2001.

[11] E. O. Ribeiro, M. Walter, M. M. Costa, R. Togawa, and G. Pappas. p2pBIOFOCO: Proposing a Peer-to-Peer System for Distributed BLAST Execution. In *10th IEEE HPCC, China*, pages 594–601, 2008.

[12] A. Rowstron and P. Druschel. *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, volume 2218, pages 329–350. Springer, 2001.

[13] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, P2P '01, pages 101–, Washington, DC, USA, 2001. IEEE Computer Society.

[14] H. Sharma and J. Odell. Event-Driven Architecture (EDA) And Its Relationship with SOA & BPM. Technical report, OMG, September 2006.

[15] D. Shi, J. Yin, Z. Wu, and J. Dong. A Peer-to-Peer Approach to Large-Scale Content-Based Publish-Subscribe. In *Proc. of the 2006 IEEE/WIC/ACM WI-IATW*, pages 172–175, USA, 2006. IEEE Computer Society.

[16] G. Spanoudakis and K. Androutsopoulos. Monitoring security and dependability in mobile p2p systems. In *Proc. of The 3rd International Conference for Internet Technology and Secured Transactions*, ICITST '08. IEEE Computer Society, 2008.

[17] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM*, volume 31, pages 149–160, New York, NY, USA, October 2001. ACM.

[18] L. Stout, M. A. Murphy, and S. Goasguen. Kestrel: an XMPP-based Framework for Many Task Computing Applications. In *Proc. of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 11:1–11:6, New York, NY, USA, 2009. ACM.

[19] W. Yeager and J. Williams. Secure peer-to-peer networking: The jxta example. *IT Professional*, 4:53–57, 2002.