BISSA: Empowering Web-Gadget Communication with Tuple Spaces

Udayanga S. Wickramasinghe† Charith D. Wickramarachchi‡ Pradeep R. Fernando‡
Dulanjanie Sumanasena† Sanjiva Weerawarana‡ Srinath Perera‡

ABSTRACT
Modern web pages are not just static plain HTML files. With the invention of browser scripting languages such as JavaScript, web pages have come alive. Still, there is no such mechanism for unified communication between browser applications that use scripting languages. We propose BISSA, a communication model which provides a unified & time-decoupled communication platform based on tuple spaces for browser applications. Our proposed solution consists of an in-browser tuple space implementation & scalable and distributed peer-to-peer global tuple space implementation which can act standalone as well as collaborate with each other seamlessly.

The in-browser tuple space provides a solid communication infrastructure for web gadgets. This paradigm of communication is further enforced by the integration of the in-browser tuple space with the peer-to-peer global tuple space, effectively allowing web-gadgets to contribute/co-ordinate with the underlying computing infrastructure.

In this paper we state & present our implementation methodology, and show how browser applications can use this infrastructure as an inter-gadget communication solution, storage platform for application generated data and as a middleware to develop web-based applications that brings the computation power of browser in to the grid.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architecture—patterns, data abstraction

General Terms
Design, Experimentation

Keywords
Tuple Spaces, Web-Gadgets, DHT, Peer to Peer

1. INTRODUCTION
Web gadgets are independent entities in a web-page. Each gadget can be considered as a separate HTML page, rendered by a gadget server. The concept of web-gadgets has revolutionized the World Wide Web (WWW), as it brings the component based design in to a HTML page. Yet the platform support to facilitate the communication between these components is minimal (discusses further in the coming sections), which makes web-gadgets to appear as isolated entities.

The tuple space model provides a convenient programming model for communication between applications, compared to the general message passing models. In message passing models, parties that communicate with each other have to be alive at the same time. The advantage of tuple space communication is that the parties that communicate using the tuple space need not be connected at the same time to engage in the communication. Furthermore the processes can coexist in the space/network without knowing each other’s existence. These two factors sum up to making tuple space model time and space decoupled, a feature very few parallel/distributed processing architectures possesses [2, 3, 18].

We believe that we can provide a more sophisticated but programmer friendly communication infrastructure for web-gadgets using tuple spaces concept. Here describe a tuple space based communication infrastructure which provides,

- Communication between web-gadgets within a browser instance: The web gadgets may reside in same window or in two different windows.
- Communication between web-gadgets, which resides on different browser instances: Here the browser instances may run in a shared physical computer or two distinct computers.

Our approach provides tuple space implementation with two levels, identified as in-browser tuple space & a global tuple space. The BISSA browser tuple space is based on JavaScript and provides a JavaScript based API to access the space while BISSA global space is implemented as a distributed peer-to-peer system using a distributed hash table (DHT). The API enables applications to consume standalone local tuple space (residing within the browser memory) or to seamlessly integrate with the distributed peer to peer tuple space. We used Apache Shindig [17] as our web-gadget container & wrapped our in-browser tuple space as a Shindig feature to test the behaviour.

This paper focuses on the architectural & Implementation level aspects of the presented in-browser tuple space & the supporting peer-to-peer global tuple space. The paper starts with a related work section followed by small overview of the proposed system in Section 3. Section 4 gives a brief description on the global tuple
space implementation concepts while Section 5 covers design & implementation aspects of the in-browser tuple space in detail. In Section 6 we have presented our results & analysis. Section 7 discusses possible use-cases & desirable properties of the proposed system & Section 7 concludes.

2. RELATED WORK
The concept of tuple space emerged in the 1980 s’ with the invent of Linda programming language by David Gelernter and Nicholas Carriero at the Yale University. In their implementation they supported four basic primitives, namely in, out, rd & eval [3]. The model gain its popularity due its simplicity, features of orthogonality and the spatial & temporal decoupling that it provides to the concurrent processes [18]. Since then there has been lot of researches based on tuple spaces over the years & we can find various tuple space implementations targeting different environments [8, 12] & use cases [2, 4, 5].

Duple is a P2P based tuple space implementation that has characteristics similar to BISSA P2P implementation. Both BISSA & Duplex use Pastry as its routing overlay but BISSA focuses on eventual consistency model that enhances scalability while Duplex relies on transactions to maintain the consistency [11]. Further, BISSA P2P implementation make use of PAST [15, 7] for its tuple storage purposes after extending it to our specific needs.

Web-gadgets are HTML/JavaScript/CSS content that operates as independent entities inside a web-application. They are specified in a XML declaration file conforming to a specification, which makes sure its compatibility among different environments. It is a gadget container’s responsibility to render gadget layouts and controls, process metadata, user preferences etc and deploy “features”[17] as specified by declarative xml syntax of a gadget.

Apache Shindig [17] is an OpenSocial [13] compliant gadget container, providing the infrastructure needed to host and access XML based gadgets. Shindig comprises of a JavaScript based gadget container which provides core gadget functionalities such as gadget communication, layout, security etc., a gadget rendering server which outputs XML based gadgets into JavaScript and HTML and an OpenSocial container and Data server for OpenSocial based gadget communication.

Shindig provides different capabilities to gadgets it renders, through a mechanism called “Shindig Features”. Shindig features are JavaScript libraries with some useful functionality provided either to gadgets, containers or both at the deployment time.

Shindig Pub sub, a feature intended to facilitate inter gadget communication, provides a communication infrastructure through a topic based publish/subscribe paradigm for xml based gadgets. BISSA provides a tuple space based shared memory for web gadgets so that while supporting all communication scenarios that can be achieved with the shindig pub sub feature, BISSA can also act as a shared memory for the browser gadgets.

SQUIRREL [10] is a decentralized, peer-to-peer web cache that makes use of Pastry as its routing overlay. It enables the web browsers on desktop machines to share their local caches. A caching solution implemented using BISSA will have the similar characteristics, but will differ in its implementation since the objects have to be encoded as tuples. SQUIRREL act as a cache for browsers in a LAN environment, while the web-application themselves can use BISSA as their distributed caching solution for application generated data. The consistent API across in-browser tuple space & the global tuple space make sure that the peer to peer nature of the space remains transparent to the applications.

SETI (Search for extraterrestrial intelligence) is a scientific area whose goal is to detect intelligence outside the earth. In their effort they process narrow bandwidth radio signals which are gathered using radio telescopes. The processing of data requires large amounts of computing power. As a solution they started the SETI@home[16] project which uses community powered virtual supercomputer to process their huge sets of data. In order to contribute to the project volunteers have to download a client software which is in the form of screen saver. BISSA middleware can be used to power up above kind of effort. The usage of Bissa will extend the idea of community contributed processing power in to the browser space; where web-application themselves act as nodes in the computing grid.

3. THE BIG PICTURE
BISSA creates an infrastructure that expands over intra-nets, possibly over the Internet on different platforms where each node is contributing to the distributed memory of the tuple space. With local and global space integration in place we can see some of these nodes expose BISSA global tuple space as a web service, enabling independent browser instances to act as clients, providing them the ability to access global space as well as their own local ones. This is shown in the following diagram, figure 1.

![Figure 1: High level deployment view of BISSA](image)

In turn these local tuple space instances would provide the inter-gadget communication infrastructure for web based gadgets so that multiple gadgets participating can form very robust and powerful web based applications integrated across a geographically dispersed area. The fact that now these gadgets are effectively connected across a scalable global tuple space provides them obvious advantages such as high level of persistence and scalability. Additionally BISSA enables different clients other than browsers to connect to global tuple space, making it accessible to any web service enabled or REST based client such as mobile devices, java clients, etc.

4. BISSA PEER TO PEER SPACE
BISSA global space is a distributed, location transparent tuple space & is based on peer to peer paradigm. It uses Pastry [14], a
distributed hash table based routing overlay for routing the content based on their hash values. The space is created by the nodes which get connected in to the peer to peer network. Applications will get the abstraction of a local tuple space, while in reality the tuples get stored in a distributed manner within the P2P network. In our implementation we used PAST [15] an archival storage facility after extending it for our specific needs.

4.1 Primitives Supported
The peer to peer global tuple space supports three standard operations named read, write & take. The read & write operations are available in both synchronous & asynchronous flavours while take operation is a synchronous only operation. Apart from the three standard operations, the space also provides a subscription mechanism based on tuple templates.

The following section describes the implementation details of the operations at a very high level.

4.1.1 Write Operation
The write operation inserts a tuple in to the space. Underlying actions such as distributing & replicating data will be invisible to the user. At a very high level we can identify two main phases associated with the write operation.
1. Write tuple as a DHT element with an associated hash calculated for it.
2. Update the Index files which are distributed in the system.

4.1.2 Read Operation
Read operation is the simplest of all operations, where it returns the tuples that matches the given template. As an example <hello,world> & <hello,bissa> will be returned if the user reads the space using the template <hello,???> given the fact that above tuples are already in the space.

When a user queries the space using a template, it first generate the hash of the given tuple template. Then it looks up the index structure associated with the generated hash value. The index structure contains all the hash ids of the tuples that matches the queried template. Based on the retrieved hash values from the index structure it then locate & retrieve the set of tuple associated with the given template.

In our implementation we support two flavours of the read operation,
1. A blocking read where user call is blocked till all the tuples are received.
2. A non-blocking read: receiving tuples will be passed on to the user at the time they receive thus users can avoid waiting till the availability of all the results.

4.1.3 Take Operation
The take operation differs from the read operation, that it removes all the tuples from the space that matches the given template. Removing tuple instance that has been replicated among the nodes of a peer-to-peer system raise some serious issues.

Take operation consists of three steps that is designed to bring eventual consistency in to the system,
1. The retrieval of the tuple from the peer-to-peer system that matches the given template.
2. Deletion of the tuple content from the peer-to-peer system.
3. Updating the indexes that carry wild card info of the deleted tuple.

5. BISSA IN-BROWSER TUPLE SPACE
BISSA browser tuple space resides in a local browser instance memory and integrates into the global tuple space. However depending on application programmers’ choice BISSA browser space can remain as an independent tuple space residing only in browser memory or as a fully integrated tuple space that is being synchronized with global tuple space environment as well. Local tuple space API is consistent with the global tuple space; hence all the primitives available in global space are true for local space as well.

Local tuple space can be considered as a combination of a tuple pool (TP) that keeps track of tuples in local space and a hash table instance (TPHT- tuples to processes hash table) that associates tuples with the respective local processes which involve in the tuple exchange. Whenever a tuple is written into the local space, the tuple pool is updated accordingly and relevant processes are notified of the availability of tuples using the hash table. If a local process wants to query a tuple T or a specific template T_n (a tuple with wild card entries in it), local tuple space manager will match T or T_n with the tuples set {T_1, T_2 ..., T_n}in TP. A special case only occurs when matching a tuple set with a given template T_e.

5.1 Integration
BISSA provides access to global tuple space for users who work within local browser instances by providing a uniform interface that can seamlessly communicate with both local and global tuple space memory easily and flexibly. Integration approach for BISSA infrastructure for local browser clients is through web-services that wrap BISSA runtime instances. Prospective clients who wish to query tuples distributed in the global space (i.e.-: in different geographic localities) can use the JavaScript API provided by local tuple space runtime inside the browser instance. This mechanism also provides an efficient way of communicating between two or more browsers located in two different machines or even within two browser windows in the same machine.

BISSA local tuple space connects to a BISSA web-service node via an Ajax connectivity layer. The browser tuple space will synchronize its localized tuple content during each connection session as required. As a result any process those queries for a certain tuple or a tuple template will be delivered all the conforming tuple content from the local tuple space as well as global space. This model is pull based since browser tuple space itself is responsible for pulling tuple content/data out from the globally scalable tuple space. The following diagram, figure 2 describes this overall model of integration.

Fig 2: Integration model of BISSA
5.2 Messaging Layer
Local space queries for tuple content provided by the API are differentiated into two categories, synchronous and asynchronous. Synchronous method calls would wait until global web-service enabled global node replies for the tuple query while asynchronous calls would notify the node and resume execution. Users have the choice of adapting any of these variations depending on the requirement or performance considerations.

Sometimes it may be important that the browser application relieves the connection burden on global tuple space and resume the local work it has been engaged in to increase performance of the application. BISSA asynchronous connectivity supports this requirement by temporary delegating tuple queries to global layer and letting it do the work for you.

On the other hand, sometimes applications may want to constantly query content n a specific template. Application programs can use subscribe option in these scenarios, to bind into global space for specific tuple queries. BISSA will take care of the periodic querying on global space for the respective tuples/templates and notifying the relevant processes which subscribed/bind to them.

5.3 Web-Gadget Communication
Usually web application integration spanning across the internet tend to be ugly and highly complex and coupled with the increase in number of integration units. This inherent nature of web applications can be exploited by BISSA to provide an ideal way of inter-communication, since this is exactly and fundamentally what BISSA tries to resolve, “Providing a shared memory abstraction for naturally tightly coupled set of processes/entities”. So in-effect communication between these set of application entities/components can be facilitated through writing and retrieving tuples through BISSA (shared) tuple space, greatly reducing complexity or coupling between the components involved.

In implementation aspects this concept of using BISSA as an inter-communication framework has presented us with several challenges. That is, where/in which aspect of web we should be implementing such a system? And how exactly it should be implemented to achieve an almost unified way of inter-communication? The answer lies in web gadgets. Specified by xml syntax gadgets are considered independent entities even when they are contained within a single web page. So as per the answer to our first question web gadgets seems to be a very good choice due to their independent and highly coupled nature. For example BISSA can present a shared channel for gadgets operating inside a web page or set of gadgets integrated across a network/internet, reducing complex interactions which otherwise would have required.

Web gadgets also seem to be the ideal answer to our second question as well. The solution lies in “features” which are a standard compliant way to implement specific services for web gadgets. This is in the sense that, BISSA browser/local space runtime can be wrapped into a gadget feature [17], to be deployed on gadget servers at runtime. Therefore gadgets can be deployed in different gadget servers (provided we have a BISSA feature implementation for each different gadget server i.e.: Shindig, iGoogle, etc) exposing them to specific BISSA runtime libraries creating a unified infrastructure for inter-gadget communication. Using this kind of BISSA enabled feature, gadget developers could use the functionality described above to facilitate communication between gadgets located in a single browser window (is: using local space API) or even between different gadgets located in separate servers.

Figure 3, is high-level overview of how a BISSA enabled feature can be deployed on a gadget server.

![Fig 3: Overview of the BISSA feature implementation](image)

5.4 Supported API
BISSA local space provides a clean and consistent API to support inter gadget communication. Although related to generic tuple space API, functions supported makes into two categories.

a. Local tuple space API
b. Global tuple space integrated API

This differentiation gives application developers a clear separation of concerns whether to use a standalone local tuple space or to get exposed to the distributed tuple Space.

5.4.1 Local Tuple Space API
Following are the supported local space primitives.

a. bissa.read(t_template(callBack,subscribe))–read a tuple from the space relevant to the given template.
b. bissa.take(t_template,callbck,subscribe)–read the tuple and remove it from the space.
c. bissa.put(tuple)–insert a tuple into the space, subscribed users will be notified if requested tuple is being inserted.

5.4.2 Global Tuple Space API
Following are the Global Space primitives supported within browser Space integration,

da. bissa.read_global–read a tuple from the global space relevant to the given template
e. bissa.take_global–read the tuple and remove it from the global and local space.
f. bissa.put_global(tuple)–insert a tuple into the space, and attempts to update tuple content globally.

6. RESULTS & ANALYSIS
Upon successful completion of the middleware implementation we performed a scalability test and a latency test to measure the behavior and performance of our peer-to-peer global tuple space. Furthermore we carried out a test to measure the operational
latency of the browser tuple space. The results and analysis of this test are as follows.

6.1 Scalability Test
We used Monte Carlo simulation [13] as our benchmarking strategy. It is an embarrassingly parallel algorithm, which involves independently executing disconnected components [14]. We used BISSA as the middleware to share tasks between processors and finally to aggregate the results.

We carried out these tests in a controlled lab environment with machines with same configurations (Intel core2duo processors, 1 GB RAM, Fedora OS) and did the scalability test with up to 35 computers. Figure 4 shows the results obtained for the scale test with the increasing number of nodes.

6.2 Analysis of the Scalability Test
As shown in figure 4 we got an almost ideal parallelism with the scaling of the system. Since we tested only with 35 nodes we can’t claim that our implementation is massively scalable. But with these results we can say with confidence that our system is a reasonably scalable one. The results show a slight performance lag when the node count is above 30. This is due to the amount of work per worker node is becoming smaller & as a result the communication latency comes into effect.

6.3 Latency Test
One other test performed on the BISSA peer-to-peer space was the operational latency test. It was carried out to measure the time taken for an operation to complete. Here we measured the completion time of the read( ) & put( ) operations on the BISSA global tuple space with an increasing number of nodes in the network.

6.4 Analysis of the Latency Test
According to the figure 5 it is apparent that the latency of read and put operations are hardly affected by the increasing number of nodes connected to the space. The result shows that the operational latency is hardly affected by the number of processes involved in reading and writing tuples to the space. According to the pastry algorithm the worst case the message delivery latency is a function of number of nodes in the peer to peer network. Although our result shows a minimal variation in latencies with the number of nodes introduced, we must expect the latency of the operations to be increased when the number of nodes is large.

The research conducted by Kato and Kamia [12] shows that the messaging latency of pastry is quite good up to 800 nodes. Hence we can argue that, BISSA being built on top of pastry and having showed good results in the tests we have conducted, would continue to perform well for an even larger number of nodes.

6.5 Browser Space Operational Latency
To measure the performance of the operations on the global tuple space by the browser tuple space; we carried out latency tests for read, take and write operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read( )</td>
<td>115</td>
</tr>
<tr>
<td>Take( )</td>
<td>332</td>
</tr>
<tr>
<td>Write( )</td>
<td>98</td>
</tr>
</tbody>
</table>

Major portion of this time is due to web services access time from the browser. Thus the resultant time is heavily dependent on the users’ network environment.

6.6 Analysis
It is clear that the operational latency of the browser space global operations is significantly higher than the peer to peer space operations. Since this latency is mainly due to the network access time. Hence we can conclude that the operational latency
behaviour browser space global operations with an increasing number of nodes in the peer-to-peer space will remain similar to Figure 5.

7. CASE STUDIES

7.1 Role of web browser in distributed processing

The world is moving from standard desktop applications to web-based applications. The modern operating systems such as Chrome OS establish this concept further. With browser becomes more prominent in the computing market, we should explore the possibilities of getting the processing power of the browser in to distributed processing environments.

There are popular volunteer computing platforms such as BOINC [1], which powers huge data-processing applications such as SETI@home [16]. In order to contribute to the grid, people have to install a certain client on their computing device. While it is acceptable for most of the situations, there can be situations where installing an third party software client is not secure enough or may be someone’s computing device entirely based on the browser. But still they may want to contribute to the grid.

Such applications can make use of BISSA as their underlying communication middleware. The peer to peer global tuple space will act as the shared memory between all the browser instances. JavaScript processing clients can fetch data from the space, process them & write back to the global space. Standard desktop applications may use BISSA P2P Java library to contribute to the same network, or else they may contribute via the provided web-service.

7.2 Co-ordination among Gadget Based Applications

The BISSA communication platform can be easily used for co-ordination between web-gadgets which reside in a web page. For an example consider two web-gadgets that displays a world map & the weather data of a region respectively. But what if I want to get weather details of a particular region, by clicking on that region in the world map?

With BISSA in-browser tuple space, web-gadgets can communicate/co-ordinate with each other without much of a hassle. Because of the whiteboard pattern based communication paradigm the data is available to any gadget, whenever they want, and avoiding complex synchronization issues.

Further, the seamless integration of BISSA in-browser tuple space with BISSA global (P2P) tuple space allows web gadgets to co-ordinate with each other while residing in two different machines. As an example when you click on a region in the world map web gadget, the weather details of that region will be displayed on the weather data web-gadget which resides on your friends’ browser.

7.3 Highly Scalable and Available Data Storage for Web Applications

The peer to peer global space that integrates in to the in-browser tuple space gives immense power to the browser based applications such as web gadgets. The global space can be act as an alternate data storage facility for web-applications.

For an example think about an organizational environment where hundreds of people use remotely hosted web-app. It is desirable to access the web-application in offline mode whenever needed. In order to do that we need temporary storage mechanism similar to Google gears [9].

A peer to peer global tuple space deployed over the organizational environment may solve this issue effectively. Here the deployed P2P global space acts as a highly available & scalable intermediate data storage facility for the web-applications, allowing them not to reach the primary backend server whenever possible, thus improving the performance & flexibility.

7.4 As a Pub sub Framework for Web Applications

The inherent nature of the tuple spaces provides an ideal pub sub framework for browser applications. The alternative approach would be to implement a pub sub manager within the browser, allowing applications to publish & subscribe for the data based on topics. While this approach works fine in a static environment, there can be situations where applications do not get loaded at the same time. What if the application interested in a certain set of data that was published already, before the interested application comes alive?

In-order to solve the issue, what effectively need is data persistency. The pub sub framework implemented using a tuple space requires no subscription manager & allows applications to publish & make use of the data, without loss of data due to synchronization issues.

7.5 Using BISSA as a Cache

The availability of in-browser tuple space enables web-applications to use that facility as a cooperate cache for the frequently used browser data. Any complex data component can be break down into an array of primitives & can be stored as a tuple. Thus the in-browser tuple space can be effectively use as a browser cache & the in-memory nature of the in-browser tuple space is a favourable characteristic for the proposed use case.

It is possible to use BISSA as a distributed web caching solution in an organizational environment. The caching can make use of the in-browser tuple space as well as P2P tuple space to store the data. The peer to peer nature of the BISSA global space negates the need of a dedicated caching node thus making such solution ideal for a small organizational environment.

8. CONCLUSION

Here we presented the design & implementation details of BISSA, a middleware infrastructure based on tuple spaces for browser applications.

The middleware implementation consists of two main components;

1. An in-browser tuple space.
2. A peer to peer global tuple space.

Browser based tuple space can act as a standalone middle-ware & supports inter gadget communication/ inter java script component communication within the browser. Peer to peer tuple space can be used by Java application clients as a distributed shared memory.
The integration of the two components with each other results in a powerful middleware platform for JavaScript based browser applications such as web-gadgets. The web-applications can use the middleware as inter-gadget communication infrastructure, data storage facility for java script based applications, distributed shared memory for parallel processing, etc based on their requirement.

9. REFERENCES


[17] SETI@Home Project: What is SETI @ HOME, 2010, Retrieved April 2010, from University of California SETI@Home project: http://setiathome.berkeley.edu/
