Using a RESTful messaging and registry system to support a range a distributed applications

Mark Baker, Garry Smith, Matthew Grove, Rahim Lakhoo, Hugo Mills, and Carl Albing,
School of Systems Engineering, University of Reading, Reading, UK
e-mail: mark.baker@computer.org

Abstract — Tycho was conceived in 2003 in response to a need by the GridRM [1] resource-monitoring project for a "light-weight", scalable and easy to use wide-area distributed registry and messaging system. Since Tycho's first release in 2006 a number of modifications have been made to the system to make it easier to use and more flexible. Since its inception, Tycho has been utilised across a number of application domains including wide-area resource monitoring, distributed queries across archival databases, providing services for the nodes of a Cray supercomputer, and as a system for transferring multi-terabyte scientific datasets across the Internet. This paper provides an overview of the initial Tycho system, describes a number of applications that utilise Tycho, discusses a number of new utilities, and how the Tycho infrastructure has evolved in response to experience of building applications with it.

Keywords- RESTful, HTTP, PUT/GET, Transactions, Web 2.0

I. INTRODUCTION

The Tycho [2] project's focus is on the design and development of a system for binding together distributed applications with the aid of a combined messaging and registry system. Tycho provides application developers with a means of securely integrating distributed systems using a single software package. One aim is to simplify the process of assembling distributed applications by reducing the number of libraries or tools required by the application developer. A second aim is to produce a system that is more scalable and has higher performance than the combinations of registry and messaging software previously available.

Tycho's architecture is shown in Figure 1, and is described here [3]. Tycho has a service oriented architecture, and it main components are:

- Mediators allow producers and consumers to discover each other and establish remote communications.
- Consumers typically subscribe to receive information or events from producers.
- Producers gather and publish information for consumers.

The design philosophy for Tycho was to keep its core relatively small, simple and efficient, so that it has a minimal memory foot-print, is easy to install, and is capable of providing robust and reliable services. More sophisticated services can then be built on this core and are provided via libraries and tools to applications. This enables Tycho to be flexible and extensible so that it will be possible to incorporate additional features and functionality.

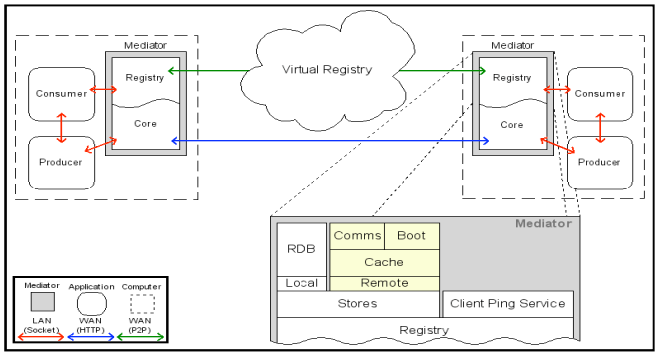

Figure 1: Tycho's Architecture

In Tycho, producers and/or consumers publish their existence in a directory service known as the Virtual Registry (VR). A client uses the VR to locate other clients, which act as a source or sink for the data they are interested in. The VR is a distributed peer-to-peer service provided by the network of mediators. Normally clients communicate directly, however, for clients that do not have direct access to the Internet, the mediator provides wide-area connectivity by acting as a gateway or proxy into a localised Tycho installation. The Tycho VR is made up of a collection of services that provides the management of client information and facilitates locating and querying remote Tycho installations. A consumer (client) registers with a local mediator, part of the VR, when it starts-up. The VR provides a locally unique name for each client and periodically checks registered entities to ensure their liveliness, removing stale entries if necessary. Tycho's architecture is designed to support both encryption and access control to provide a secure environment. Encryption is provided at the transport handler level using SSL to encrypt messages sent via the HTTP and Socket handlers.

II. TYCHO IMPLEMENTATION

Tycho's implementation is based on REST [4] in order to provide standardised services without the need to follow long and complicated Web Services or Grid standards that were and are constantly in flux. Tycho was designed to be easy to deploy and as a result only requires an installed Java Virtual Machine (JVM) in order to execute the single Java JAR file that consists of the complete Tycho

978-0-7695-3680-4/09 $25.00 © 2009 IEEE
DOI 10.1109/ISPDC.2009.17

download (all Tycho dependencies such as the mediator's internal database and communications libraries for the VR are included within this file).

Tycho was intentionally designed around a minimal core that only provides essential services. The idea is that the core should be fast and unencumbered by features that are not essential to Tycho's roles as a distributed registry and messaging system. This approach differs from the method used by other middleware such as NaradaBrokering (NB) [5] and the Globus Monitoring and Discovery System [7] where new functionality is typically added to the core implementation. In Tycho, new functionality is packaged as optional utilities that sit above the core services.

Since synchronous communications typically have a higher overhead than asynchronous (due to the need to track messages and the time spent blocking and waiting for responses) a decision was made early in Tycho's development that its core should only support asynchronous communication patterns. This means that applications must either be written following an asynchronous programming model, or utilise a higher-level blocking API potentially implemented as a Tycho utility.

Here we briefly describe the main methods of the Client API:

- Class *TychoConnector*: provides the core client functionality. It contains fourteen methods that provide a high-level API for the Tycho functionality. The API handles interaction with other clients and the Tycho mediator transparently. For example, to perform a distributed Tycho query across the entire VR a single method is used with one parameter.
- Class *EmbeddedMediator*: starts an instance of a Tycho mediator within the same JVM as the client. It provides an additional three methods to retrieve settings from the mediator (such as its URL) and allows the verbosity of the debug output to be set.
- Class *Message*: contains fourteen methods for manipulating Tycho messages. It provides a high-level interface for users to read and alter parts of a binary message without having to directly manipulate the binary data.
- Finally a single Java interface is provided - Interface *IeventInterface*: defines five methods that are implemented by all Tycho clients. It allows asynchronous messages to be delivered to the client.

Tycho provides a plug-in mechanism that allows different technologies to be used for the VR communications. The reference implementation provided HTTPS and Internet Relay Chat (IRC) [8] support. IRC networks provide a communications overlay that is configured to provide fault tolerant messaging. Tycho allows applications developers to make use of this existing Internet infrastructure without the need to deploy their own messaging services, daemons, or even change the configuration of Tycho from the default options.

When using a distributed registry to store data for an application, one task is to configure the registry to handle the necessary data. To configure Globus MDS and the Relational Grid Monitoring Architecture (R-GMA) [9] to accept user-defined application specific data, all instances of R-GMA or MDS must be configured with the same data schema. In contrast, Tycho allows the clients to dynamically describe the data they publish into the VR by using the schema field, which means that VRs do not need to be reconfigured. Both the automatic configuration and the ability to dynamically publish ad-hoc information into the VR, this reduces the administrative overheads of deploying and using Tycho, compared to these other systems.

Unlike related systems, such as NB, MDS, R-GMA, Jini [10] and Apache Axis [11], Tycho is distributed with security enabled by default and does not require the developer to install additional packages. The reference implementation of Tycho supports transport-level encryption using HTTPS and access control (using MD5 hashes), which prevents access to the Virtual Registry. Furthermore, these systems have multiple software dependencies that must be satisfied as part of the installation process; Tycho on the other hand only requires a JVM to operate.

Related systems require some manual configuration before they could be bootstrapped. They also need additional effort to arrange their components into a scalable hierarchy. The Tycho mediator automatically discovers and connects to other Tycho instances by using the bootstrapping functionality of the Virtual Registry (VR)-interconnects.

Various performance tests have been made to measured registry's performance and capability of Tycho, against MDS, and R-GMA. This revealed that Tycho had better performance than the other systems. During these tests the other systems failed due to memory management issues. As part of a peer-to-peer file-sharing test, Tycho successfully transferred files of up to 80 Gbytes to 60 peers.

III. Tycho Utilities and Applications

Since Tycho was first released, a number of updates to the software have been made to make it more functional, so that it can support a greater number of distributed applications. This section first briefly describes the range of applications that use Tycho, and then we discuss extensions to Tycho that have been made to better support a range of applications.

A. XDB

The UK JISC-funded VERA project [12] is investigating the development of a virtual environment for research in archaeology. The project is based around the archaeological excavation of Silchester, a Roman town that was abandoned in the fifth century. The excavation has been running for 12 years, and has accumulated a large

database of information, stored in the IADB (Integrated Archaeological Database). One challenge within the project was to provide integrated search facilities across multiple archival databases. The test system searches for records from the Silchester IADB, held in an ordinary relational database, and a classics-oriented collection of Roman-era inscriptions found in Vindolanda database [13], which is based on a RDF store. The cross-database search engine, XDB [14], see Figure 2, was developed to investigate some of the issues arising from searching across multiple and highly disparate databases.

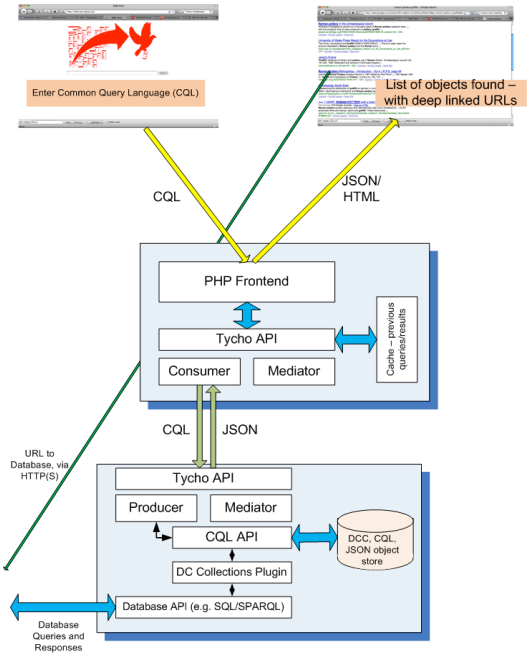

Figure 2: The XDB Architecture

B. *DNWay*

The DNway project is attempting to create a generic framework that uses a master-worker paradigm to distribute work (idempotent tasks) across the computational resources of very large supercomputers and clusters. DNway, shown in Figure 3, provides immediate, not queued, access to compute cycles and therefore must be:

- Adaptable - using processors and networks of varying speed,
- Robust - adapt to changing response times, including the failure of remote workers,
- Accessible through firewalls;
- Able to cope with different network topographies.

In order to use DNway, the "work units" that are to be distributed must be defined, and the logic that will be used to process each unit of work must be written. Resource discovery, work distribution, result delivery, timeouts, and retry mechanisms are built on top of Tycho.

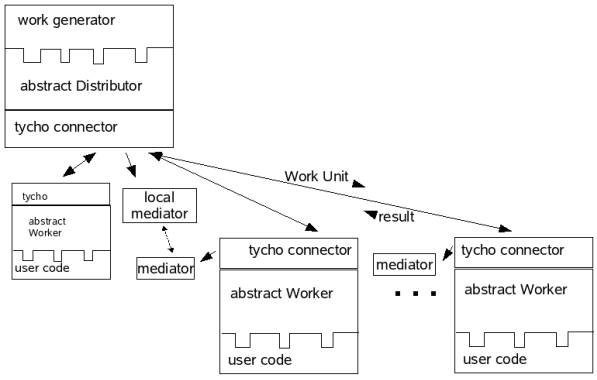

Figure 3: The DNway Architecture

C. *Necho*

The Necho project [15] is creating a multi-tiered peer-to-peer system, which is akin to BitTorrent, for distributing multi-terabyte scientific datasets across the Internet. The concepts for this project first appeared when working with the Sloan Digital Sky Survey [16], where it was necessary to split the original dataset up and use a modified version of WGET [17] to download and update the database.

The Necho architecture, shown in Figure 4, consists of a hierarchical Peer-to-Peer (P2P) system that is based around shared-portal services and unique peers donated by participating individuals and organisations. The goal of the project is to combine P2P, volunteer computing and social networks to provide a way to distribute, contribute to, and manage very large datasets. Necho uses Tycho to distribute, index and retrieve chunks of data that comprise each overall dataset. We are currently testing Necho against other BitTorrent systems, such as Azureus [18], and our single tier version of Necho is proving to be much faster.

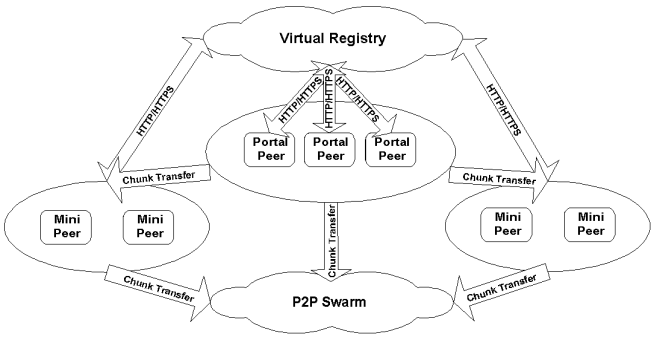

Figure 4: A Schematic of the Necho Architecture

D. *VOTechBroker*

The VOTechBroker (VOTB) [18] is a system for submitting parameter sweeps to the Grid, and other distributed resources, in a transparent way. The VOTB aims to interoperate with a wide range of job submission systems using a plug-in component, and to protect the user from middleware details (see Figure 5).

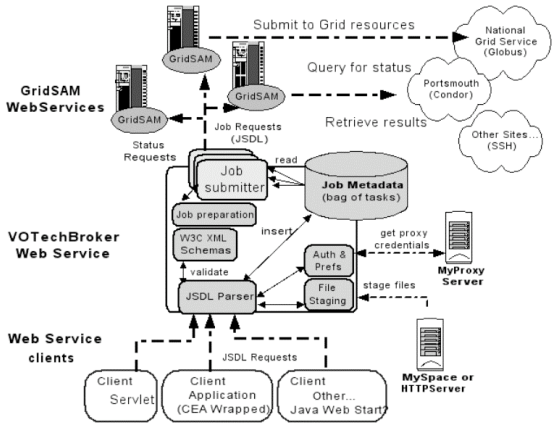

Figure 5: The VOTechBroker Architecture

A key aim of the VOTB is to support an extensible range of (Grid) middleware, hide heterogeneity, and ease the complexity associated with job submission/execution. Tycho is used to dynamically locate available computational resources at remote sites that were appropriate to a particular user's requirements (e.g. with appropriate CPU architecture, libraries and services installed, account authorisation, availability, not over loaded, and sufficient free memory).

E. GridRM

GridRM [20] is an extensible, wide-area, monitoring system that specialises in combining data from existing agents and monitoring systems so that a consistent view of the underlying resources and services can be achieved, regardless of heterogeneity. Gateways (see Figure 6) provide access to local resource information at each site. Clients connect to gateways to perform resource queries and to subscribe for events. GridRM uses Tycho in a number of ways to bind together clients and Gateways for wide-area communications, and to provide the basis of an event mechanism (both wide-area events to clients, and events from local monitoring).

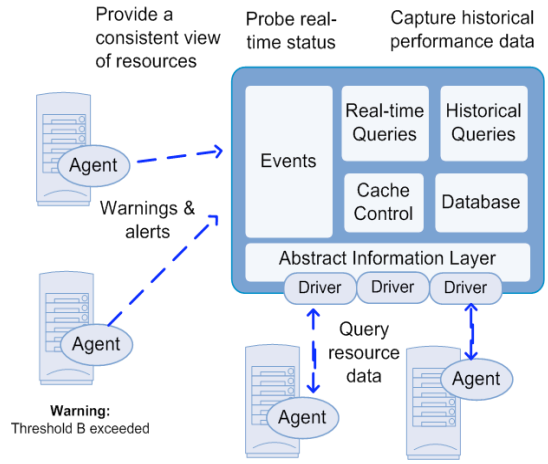

Figure 6: The GridRM Architecture

F. SORMA

In the SORMA (Self-Organizing ICT Resource Management) project, an Economically Enhanced Resource Manager (EERM) [21] exists at each resource provider's site, and acts as a centralised resource allocator that orchestrates business goals and resource requirements in order to achieve maximum economic profit and resource utilisation. The EERM's main duties (see Figure 7) include resource management, monitoring and providing standardised interfaces to resource fabrics for use from applications.

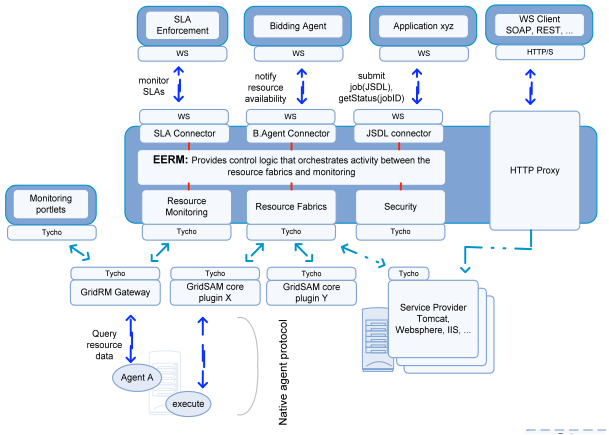

Figure 7: The SORMA EERM Architecture

The EERM utilises GridRM to obtain resource information for system and per-process monitoring in order to determine if Service Level Agreements (SLAs) have been violated. The EERM is composed as a confederation of loosely bound components for scalability and availability reasons. Tycho provides messaging and event mechanisms.

G. Slogger

Slogger [22] utilises various emerging Semantic Web technologies to gather data from heterogeneous log files generated by the various layers in a distributed system and unify them in common data store. The logs are ones generated by the operating system, middleware (e.g. Apache Tomcat or MPI) and applications themselves). Once unified, the log data can be queried and visualised in order to highlight potential problems or issues that may be occurring in the supporting software or the application itself. Slogger uses Tycho (see Figure 8) to first process, e.g. determine what data is need from the logs, and then gather data from the distributed resources and push this into a centralised RDF store. Once the data is in the store SPARQL queries are issued in order analyse the RDF log data in order to identify problem and errors in the software executing over the distributed resources.

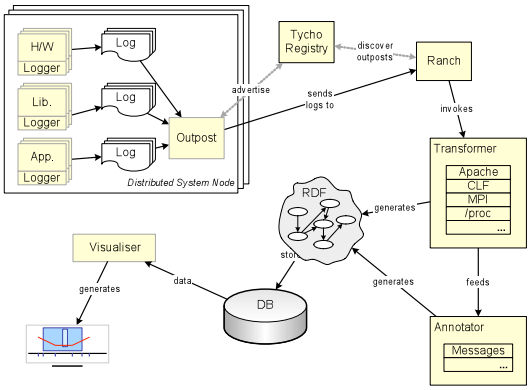

Figure 8: The Slogger Framework

H. *Map Service*

Geographical maps are used to describe the Earth's surface and its contours. These maps are hosted on servers called "map servers" [23] around the globe and can be used by scientists in many ways. For example, to find the temperature of specific area, examine the wind pressure of particular place or study the state of oceans around the world. The environmental science community lacks a searchable registry of available Map Services. As a result, scientists cannot discover the data that may be most valuable for their work or they may spend a lot of time searching for the right data. Most of the time, the scientists manage to find the service, but this may be problematic and does not give access to data needed.

The Web contains a large number of valuable environmental datasets hosted on map servers. These datasets follow different specifications such as the Open Geospatial Consortium (OGC) Web Map Service. Users access these datasets to retrieve maps of desired functionality. The demand for maps has increased in recent years, especially with the environmental science community, who are facing difficulties in finding the right map service. As a result, scientists cannot discover the data that may be useful for their research work. Often, even if the scientist finds the correct map service, the service can be unreliable and fails to provide the required data. In this project, we have developed a framework that provides a searchable database for map services.

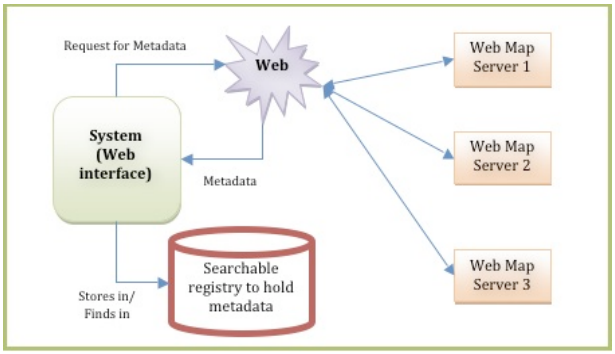

Figure 9: The Map Service Framework

Figure 9 shows the overall framework of the Map Service. This project uses Tycho to search for map services, and find those that are most appropriate, based on the search data initiated by the user. A remote producer gathers keywords and metadata from each map server, and stores this into the VR; this data is collocated with the location of the server itself. When a user wants to find map data, it sends a query via a consumer that is embedded into a Web browser; the query uses the OGC standards. The query is sent off, and first searches though the VR to find matching metadata or keywords, once this has been established, then the full OGC query is sent to the matching map server. The map server will then return the appropriate map to the client were it is displayed for the user.

I. *Web 2.0 and Portlets*

Tycho has been used with JSR-168 portlets in GridRM and SORMA. Furthermore, Web 2.0 interfaces that use Tycho to provide data to sliders have been implemented. JSR-168 portlets provide an opportunity to create user interfaces that are portable across different portal containers. We have created a number of JSR-168 portlets so that clients can remotely administer gateways and query resources from a Web browser. The portlets are currently hosted in a Gridsphere portal [31] and provide a modular approach for building a user interface; each portlet provides one type of functionality, and multiple portlets are combined (in the portal container) in order to provide the overall user interface. The portlets are categorised as those for performing gateway administration and those for querying resources and subscribing/receiving events. The portlets all utilise the client monitoring API (as a portlet service), which they use to communicate with gateways over the Global Layer. The front ends of the portlets are constructed using Java Server Pages (JSPs) that present XHTML controls and data to the client.

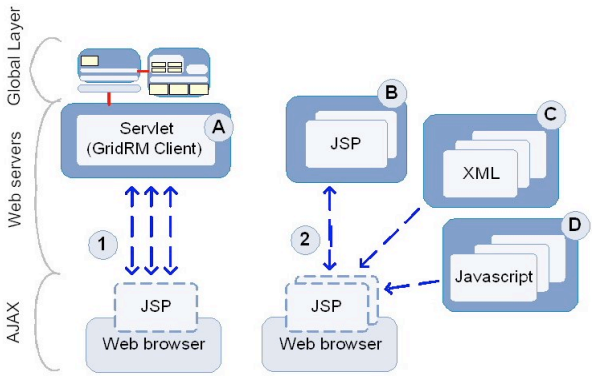

Figure 10: The Structure of the Web 2.0 Interface

Although events are passed to the portlet code (via the monitoring API) in real-time, AJAX is required to refresh data in the portlet JSPs, so that events propagate to the user asynchronously. Alternatively the user is required to interact with the portlet user interface in order to be notified of new event data (e.g. by causing the portlet to enter its `doView` mode). The structure of the Web 2.0 interface is shown on Figure 10.

Web 2.0 mashups provide an alternative way of displaying monitoring data. We have experimented with Web 2.0 to produce an interface that contains:

- Charts and gauges that display monitoring data dynamically in near real-time, for multiple resources registered with a gateway,
- A map that represents gateways according to their geographical location and displays their metadata, e.g. status, registered drivers, administrator information and network links between remote sites.
- A registry browser, that displays entries from Tycho's distributed registry in a graph and allows users to expand and collapse nodes as they browse registry meta data – this feature is useful to system developers and administrators.

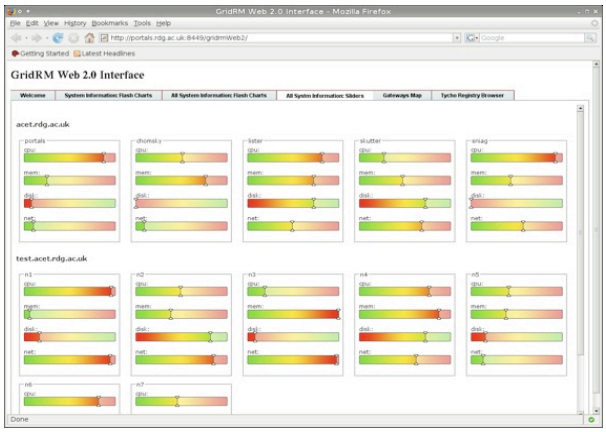

Figure 11: Slider Icons - Using JavaScript and CSS

It was interesting to compare and contrast the use of Flash-based Web 2.0 tools versus those based on simple technologies, which were less intrusive. We found that the Flash-based system contained a memory leak and would repeatedly crash the Web browser. Plus it was clear, even if the memory leak was solved, that if there were hundreds of resources being monitored, huge amount of CPU and memory would be used to monitor these resources.

As an alternative we looked at using simple slider icons, as shown in Figure 11, these were based on JavaScript and Cascading Style Sheets (CSS); they used small amount of memory and CPU, and could monitor hundreds of resource without being too intrusive.

IV. TYCHO UTILITIES

Utilities are software components that give the Tycho infrastructure greater functionality so that it can more robustly and reliably support distributed applications. In this section we describe some of the utilities that have been created.

A. Synchronous Monitoring API

By default, Tycho provides asynchronous messaging, however, many applications, such as the GridRM client requires a mix of blocking calls (e.g. for resource queries) and non-blocking calls (e.g. for event notifications). To achieve this we have created a Tycho utility to provide blocking communication operations. The Tycho method used by a client to transmit data is non-blocking and returns a unique message ID, which is used to key the semaphore into a hash table. When a message arrives at the client, the Tycho event handler passes the message content to a method that matches the response to a request ID and performs a lookup in the hash table. If a match is found, a release is called on the semaphore and the blocked call continues. If a match is not found then the Tycho message is converted into an event of a given type and sent to the client's event listener. If the client has not subscribed to receive events of the given type, then the event is silently dropped.

A client API based on Tycho has been created that consists of seventeen calls for performing common operations such as:

- Registering and un-registering interest in receiving events from particular gateways and resources,
- Registering, un-registering and listing registered resources within a particular gateway,
- Querying "core" attribute values (e.g. memory utilisation, and system load),
- Controlling "resource watches", whereby a gateway is instructed to periodically capture resource data, which it saves to its internal database for later inspection,
- Controlling "job watches", whereby a gateway instructs a capable agent to monitor process resource utilisation at defined intervals. The gateway retrieves data from the agent and stores it in an internal database for later inspection.

B. HTTP pipelining

Currently every message sent by Tycho is via a separate HTTP message. In order to optimise the communication performance of Tycho, it requires us to open multiple parallel HTTP pipes, with buffering, when sending data to a remote destination, instead of reopening the single HTTP pipe every time. This project is still underway, but effectively provides the same functionality as GlobusFTP [5]. The system being developed has one HTTP control channel, and "N" parallel HTTP channels that are used to send data between Tycho components.

C. Lightweight Transactions

This project intends to create lightweight transactions via the Tycho system so that events sent around the system will be reliable and recoverability in case a failure occurs. Although transactions are valuable and provide atomicity, persistency, and recoverability, they are not widely used in programming environments today, due to their high overheads that have been driven by the latency of saving data to disks. A major challenge in transaction-based systems is to remove disk usage from the critical path of transactions. In this project, so called "lightweight transactions" will be created. Here there will be a transaction manager, which uses main memory that decouples the performance of transactions from the disk.

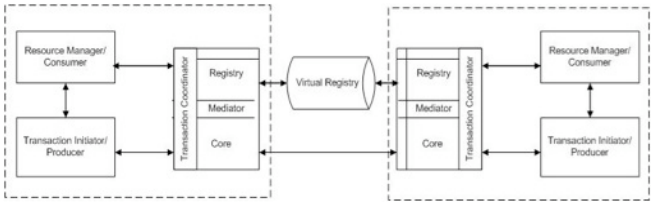

Figure 12: The RESTful Transaction System

Figure 12 shows the RESTful transaction framework embedded in Tycho, where the Transaction Coordinator is embedded inside the Tycho Mediator. In this case Resource Managers and a Transaction Initiator could be with either the Producer or Consumer. They are connected to a Tycho Mediator and thus automatically connected to the Transaction Coordinator. The Mediator manages transactions along with the core functionality of the Tycho system and returns the result to the Transaction Initiator. Implementing the Transaction Coordinator inside the Tycho Mediator creates additional functionality to the core of the system, but results in slower performance. However, it allows logging in the internal Mediator database and as a result having Log service and Transaction Coordinator in one node in the system, which creates better persistence of messages.

Once a distributed system maintains lightweight transactions, it can be used for various applications, such as computational steering or supporting events, when it is imperative to guarantee that a remote component receives the event. Principally, whenever an application is using multiple transactional, persistent resources, it may need distributed transactions.

D. Added VR Functionality

Our experience using Tycho's VR with various applications has shown us that it necessary to add greater functionality to it. The original VR was designed to store the URLs of end points (producers or consumers) and also hold various XML documents that contained useful metadata related to the end-points.

One extension for Tycho that came out of the XDB work is a results store, which manages the handling of query results from this interaction method in a thread-safe way. It does so by keeping track independently of results, which were "expected", and results, which were not. When the results store is told of a message ID that has been sent, it checks the set of "unexpected" results, and matches up the ID with any result that may have come in.

The other obvious drawback of using Tycho for the XDB system is the limited storage space and search capability within the Tycho registry. Extending the registry to support arbitrary data tables (e.g. as in Necho) would make the implementation of the XDB's "master index" functionality much easier. Going even further than that, implementing storage of RDF [29] metadata within the registry, and searching through that distributed metadata using SPARQL [30] would be highly useful too. It is not immediately obvious how a full SPARQL implementation would work to find RDF fragments split across several mediators' registry stores, but even being able to search within each mediator's store for matching fragments could offer significant benefits to the XDB.

E. Additional Caching

One way to further improve performance is altering caching in the mediator to include local data-store queries in addition to remote responses. Adding indexing to the simple store would improve its performance when searching for records. In addition, the message-passing performance could be improved by changing the socket transport handler to use thread pooling to further reduce the cost of sending messages.

V. SUMMARY AND CONCLUSIONS

Tycho is a RESTful asynchronous messaging system with an integrated peer-to-peer virtual registry. Since, Tycho was first released at the end of 2006, we have increasingly used the system to support a range of distributed applications. We have used Tycho, rather than other systems, such as the Grid, because the RESTful services provided are easy to install and use. In addition, since Tycho was first designed, the overall standards used have not changed. Tycho uses HTTP (HTTPS) and Sockets (SSL) for communications. Internally, it uses SQL as the query language and uses LDAP LDIF to mark up responses from the VR. None of these standards have changed, and it ensures that applications, based on Tycho, will continue to work for the foreseeable future.

Tycho's core is stable, but as we have pointed out in this paper, there are a number of features that need implementing to better support a wider range of applications. Some of the additional functionality needed by Tycho can be implemented by creating utilities and services on top of its generic API. An example of this is the Synchronous Monitoring API described earlier.

A. Future Work

A project is underway to develop the appropriate hardware and software to support remote monitoring of the environment via wireless sensor networks [27]. The project is using Sun SPOTs [28], which are small hardware platforms, battery operated, with a wireless device running the Squawk Java Virtual Machine (VM) without an underlying OS. This VM acts as both operating system and software application platform. We have been investigating the current software used, and we feel that using a Tycho producer/mediator on SPOT, and using HTPP communication will make the overall network more reliable and easier to program. For example, when the network starts up, each SPOT will gather data about all the SPOTs in the network. This information will be shown in the mediator and be used to calculate the optimal route to send result data back to the base station, and also, if there is a SPOT failure, it will be possible to calculate alternative routes back the base station.

Another project that we are considering is creating a system that provides "service mashups". Here Tycho components will be created, based on producers and consumers, which are registered in the VR. We will then build a graphical interface that can be used to discover and orchestra the components together. An example of a useful service mashup could be some type of workflow system, where the various Tycho components would include parts of a workflow and they can be put together in a pipeline.

VI. REFERENCES

[1] GridRM, http://gridrm.org

[2] Tycho, http://acet.rdg.ac.uk/projects/tycho/

[3] M.A. Baker, and Matthew Grove, Tycho: A Wide-area Messaging Framework with an Integrated Virtual Registry, Special Issue on Grid Technology of the International Journal of Supercomputing, (eds) George A. Gravvanis, John P. Morrison and Geoffrey C. Fox, Springer, Volume 42, pp 83-106, March 23, 2007, ISSN: 1573-0484

[4] Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, Ph.D. Thesis, University of California, Irvine, Irvine, California, 2000

[5] GridFTP, http://www.globus.org/grid_software/data/gridftp.php

[6] NaradaBrokering, http://www.naradabrokering.org/

[7] Globus, http://globus.org

[8] IRC, http://en.wikipedia.org/wiki/IRC

[9] R-GMA, http://www.r-gma.org/

[10] Jini, http://www.jini.org/

[11] Apache AXIS, http://ws.apache.org/axis/

[12] VERA, http://vera.rdg.ac.uk

[13] Vindolanda, http://vindolanda.csad.ox.ac.uk/

[14] XDB, http://xdb.vera.rdg.ac.uk/

[15] Necho, http://acet.rdg.ac.uk/projects/necho/

[16] Sloan Digital Sky Survey, http://www.sdss.org/

[17] WGET, http://www.gnu.org/software/wget/

[18] Azureus, http://azureus.sourceforge.net/

[19] VOTechBroker, https://portals.rdg.ac.uk/votb

[20] M.A. Baker and G. Smith, GridRM: An Extensible Resource Monitoring System, the proceedings of IEEE International Conference on Cluster Computing (Cluster 2003), Hong Kong, IEEE Computer Society Press, pp 207-215, 2003, ISBN 0-7695-2066-9.

[21] Self-Organizing ICT Resource Management (SORMA), http://www.iw.uni-karlsruhe.de/sorma.

[22] M.A. Baker and R. Boakes, Slogger: A Profiling and Analysis System based on Semantic Web Technologies, Special Issue of Scientific Programming on Large-Scale Programming Tools and Environments, (editors) Barbara Chapman and Dieter Kranzlmuller, International Journal of Scientific Programming, IOS Press, Vol. 16, Number 2-3, pp 183-204, 2008, ISSN 1058-9244

[23] Map Server, http://mapserver.org/

[24] JSR-000168 Portlet Specification, http://jcp.org/aboutJava/communityprocess/review/jsr168.

[25] Gridsphere Portal Framework, http://www.gridsphere.org/gridsphere/gridsphere.

[26] G.M Smith, M.A. Baker and Javier Diaz Montes, A Web 2.0 User Interface for Wide-area Resource Monitoring, 15th Mardi Gras Conference, Baton Rouge, Louisiana, 30 January - 2 February 2008, ACM SIGARCH, ISBN 978-1595930-835-0.

[27] RESN, http://acet.rdg.ac.uk/projects/resn/

[28] Sun SPOT, http://www.sunspotworld.com/

[29] RDF, http://www.w3.org/RDF/

[30] SPARQL, http://www.w3.org/TR/rdf-sparql-query/

[31] GridSphere, http://www.gridsphere.org