TriniDAT

.NET Application Data Server.

Data Mining Whitepaper.

Written & Published by GertJan de Leeuw

De Leeuw ICT | Software Development
www.DeLeeuwICT.nl
Contact: info@deleeuwict.nl

What is TriniDAT Data Application Server?

In Short: A GUI enabled webservice *and* website Development Framework written for use in .NET development projects.

Some Features of TriniDAT Application Data Server:

* Allows Extremely Rapid Application Development (RAD) with any .NET codebase attached.

* Designed for extremely fast webservice creation and code reusability.

* Spider the Internet from a server-side context using TriniDAT's native web spidering routines.

* Create websites or webservices out of thin air from existing .NET code. Run any ordinary .NET code as a website or webservice without modifications.

* Parse rich and complex media such as social content from .NET code with TriniDAT's rule-based statistical programming and filtering system.

* Write code with an easy to use & maintain event model that handles all routing of HTTP traffic for you. Events become as simple as OnGet(*myurl*) and OnPost(*myurl*)

* Works independently from Microsoft server-side technology and implementations, such as *IIS* and *ASP.NET*

And More :

* Employs a feature-rich internal object messaging system that enabled .NET classes to seperaretly interact with each other.

* Code may be webserver oriented or user interfacer oriented.The ability to mix server code with GUI's opens up a whole new paradigm of useability of server-side application programming in .NET environments.

* Features a rich text-to-speech server console with programmable host that reacts to fully programmable command handlers, like an MS-DOS command prompt.

* Dynamically install or update webservice applications.

* All Apps are naturally transferable to other servers. Professional developer can also sell their apps in the TriniDAT Appstore.

* TriniDAT App Data Server is a HTTP server that can be the technical backbone for both your online and offline project.

* With modular command handlers in TriniDAT's server console, your .NET development can reach a flexibility level comparable to a web oriented GNU/Linux scripting environment (wget etc).

/Homepage

Request Synchronization

JAlpha

...My .NET Code..

HTTP events:

.OnGET

.OnPOST

etc.

JOmega

Request Synchronization

What is hot about developing TriniDAT Data Apps in .NET?

TriniDAT chains new and existing .NET code together, forming a code chain that acts as a data CPU. This highly facciliates accuracy & efficiency with parsing of rich media content parsing (such as social media).

TriniDAT App Data server provides both webbserver and GUI functionality. Code may behave as a GUI driven client or an ordinary webserver page, or simply both. Your application is garantueed to be debugger and user friendly.

Imagine you'd need to a write a web parser that is able to rip content from specialized, non-RSS sites and push it in a database. Lets say, it would target site with highly detailed markups such as Yahoo! and CNN. The challenge in this kind of code project is managing the accurate parsing of multi-level mark-up elements such as *HTTP, HTML, Javascript and CSS*, and frankly, to not become suicidal in the process, as we did not even hint the linguistical data parsing process that would naturally be implementated after it.

Let us skip technical discussions about actual .NET code implementation for this operation and instead, let's see about bare bone minimal requirements for this kind of application. After this I'll present you how the parsing problem is handled with one solution: TriniDAT's Data App Server.

The strain of parsing multi-level media on a .NET developer is huge. A developer may ofcourse buy external code libraries to take care of each problem, but by mixing so many different codebases, a developer is garantueed to loose all control over his own code.

Workflow: Strategic Data Parsing Using TriniDAT's Framework.

1) Through class messaging features, spider Internet pages with the natively available JWebBrowser object.

2) Easily transform your harvested Internet data to statistical elements which makes it subject to TriniDAT's rule-based filter engine.

3) Optionally, one may configure specific data collection vs loose data.

4) End result: A .NET codebase is parsing the information in a highly abstract yet organized server environment, but with abstraction doubled, due to the use of .NET with the TriniDAT server environment.

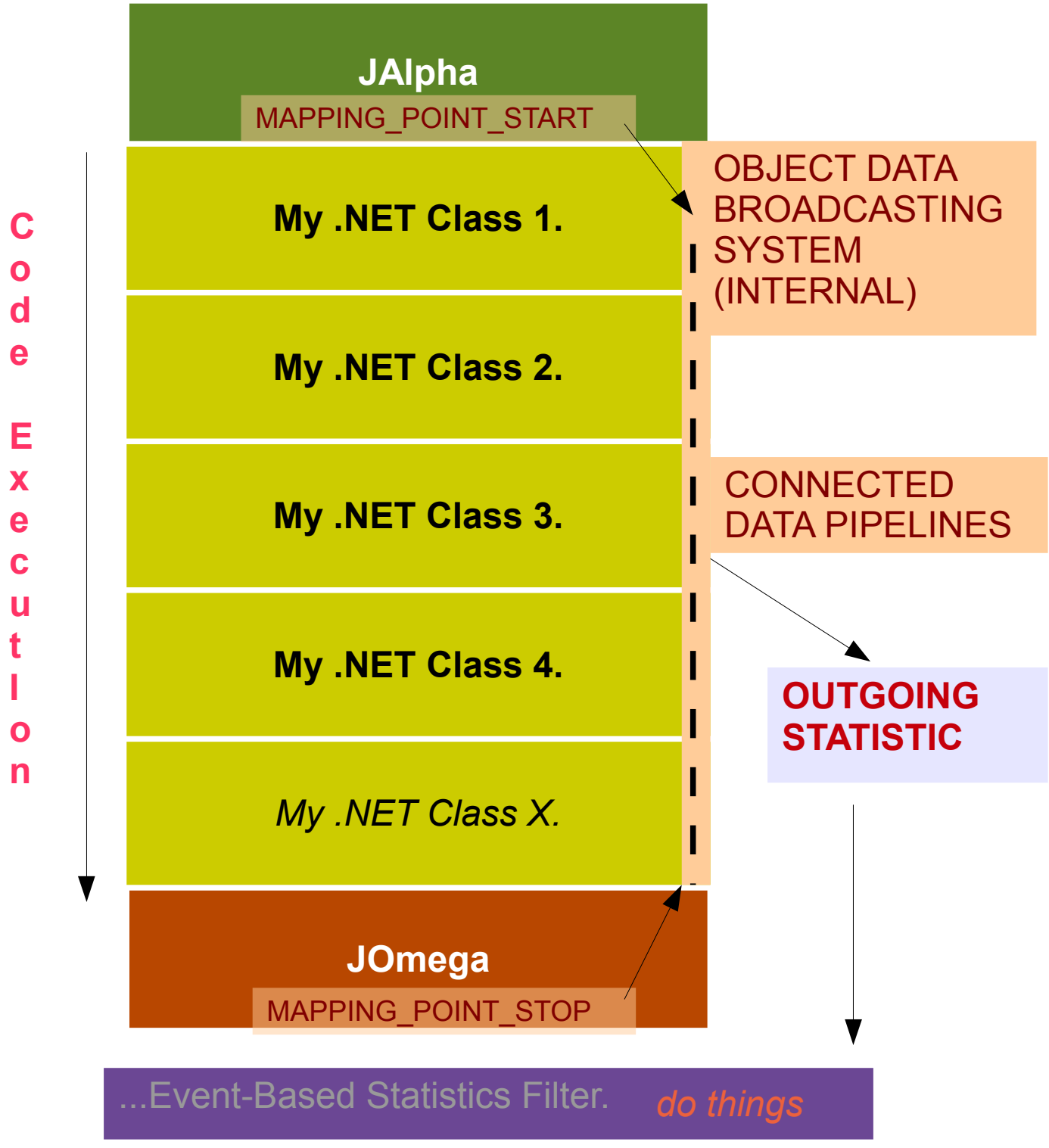
In TriniDAT Application Data Server, code dependent one manufacturer will be easily reconfigurable to make it work with code of another developer.This is so due the inherited benefits of using a indirect, yet protocolized communication messaging system. Just like forwarding an email to other persons, one only has to change the recipient fields instead editing the email's body text itself. In this respect does the internal messaging system works exactly the same as sending an e-mail.

This whole set up garantuees that one's actual .NET code is sufficiently abstract to make it even qualify for code reuse.

5) The real work: Design of data parsing strategies using a set of conditional rule filters + .NET code to trigger in the TriniDAT server context.

.NET Code Chain Trigger Using Web Browser Request.

TriniDAT Mapping Point App



Curiously, the above workflow reveals that the actual work resides in *strategical configuration of TriniDAT's rule-based statistical engine* and not in hardcore .NET development with linkage to an external code that will make external-code independence practically impossible.

Now, let's go back to the difficult technical requirements put on the example's .NET code.

Without assistance technologies one would end up with a development project that would only be specialized in the parsing of one news website type (for example, Yahoo! News or CNN.com).

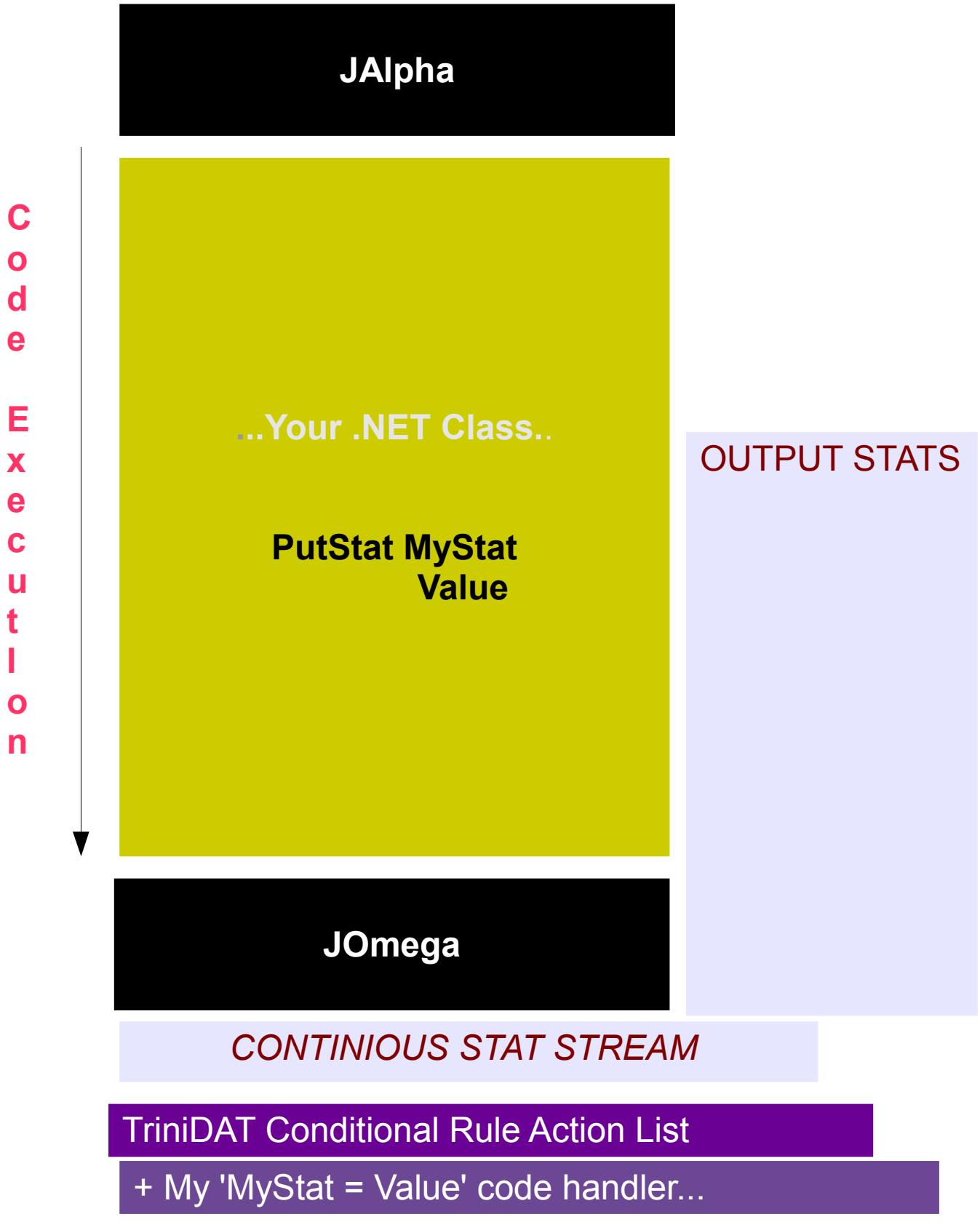
With *TriniDAT's application development mode*l you can write parsing code in specific modular blocks and share data with other modules in the webserver using an internal messaging broadcast system. It works like ordinary class methods but is strictly simplified to a message + attachment programming model, similair to ordinary internet email. This greatly improves code readability and effortlessly improves the application design.

The design of TriniDAT's infrastructure of itself promotes code reusablility for future .NET development projects.

A *TriniDAT* app developer will also be able to monetize his coding specializations by selling it along with demo application(s) in the TriniDAT Appstore, where the price is set by the code publishers.

Thus, with TriniDAT's statistical event system, a developer is able to 'drop' statistics in a server-managed data filtering stream and can then apply contextual triggers. If a new statistical item enters the data stream, TriniDAT's framework will trigger code associated with the stat's content. Therefore, this system is likeable to chain reaction of .NET assemblies ala firework that activate on HTTP requests or real-time data collectables .

HTTP://MyServer/Url



An even greater advantage of the stat filter is the fact that a data parser code handler can exclusively be allowed to wake up on a specific type of data only. This kind of 'on-demand' data parsing greatly improves perfomance and code reliability.

One such daily advantange in this kind of development environment is the efact that developers will have more time to focus on quality matters, such as quality of content rather than *software technique*. The application framework therefore effectively removes a developer from the need of logical analysis of unlogical data – which limits understanding of parsing chaotic, real-time data.

At heart, *TriniDAT App Data Server* is the powerful tool that -hypothetically- could provide a government-level data filtering system

It is a .NET development framework suitable for applications that collect realtime content from the Internet such as social media. Since TriniDAT's statistical engine utilizes realtime & in-context data parsing, it may well be used for machine interpretation of human language. *Ad Infinitum.*

Another advantage example of statistical driven event development using TriniDAT App Server: the payment module of a .NET store may drop a IP address statistical item. Extra tight security validation code may be set to wake-up if a suspicious IP address range just posted a sale, but not if the sale is from a trusted zone. This again is an example of how contextual code can greatly improves software performance Because TriniDAT App Data server is HTTP protocol oriented, any TCP/IP enabled software can make us of it's the statistical backbone.This is analogous to a Unix daemon or a Windows background service with internet connectivity configured.

As a short summary, TriniDAT application development faciliates the reuse of one-time written specialized code. After you code a parser module once, you can embed it in new TriniDAT apps that you can sell in the Appstore.

Thus one can share existing code in new product versions and varation without the overhead of revisioning systems.

“Life In Software”

Please send questions related to this white paper to: gertjan@deleeuwict.nl