Impacto de implementaciones web del patrón MVC en los requisitos de calidad percibidos

Manuel Vázquez Acosta

Universidad de las Ciencias Informáticas, Cuba

manuelva@uci.cu

RESUMEN

En este artículo se exploran dos variantes de la implementación del patrón MVC en los marcos de trabajo Web y el impacto de estas decisiones en la escalabilidad y el tiempo de respuesta percibido.

Se muestra que, en algunos contextos, una implementación del estilo Modelo-Vista-Controlador donde los conectores sean tuberías supera a las implementaciones tradicionales (llamada-respuesta) tanto en tiempo de respuesta percibido como en escalabilidad. Para esto se dan resultados de experimentos sobre dos implementaciones del patrón.

Palabras clave: arquitectura, experimentación, calidad percibida, corrutinas, patrones de diseño

ABSTRACT

This paper explores two implementations of the model-view-controller architectural style in Web frameworks, and how they affect both the perceived response time and scalability.

It is shown that in certain situations implementations of the MVC pattern which uses pipes for the connectors between components, improves over classical implementations that implements connector with a call and return pattern. An experiment has been carried to show these results.

Keywords: experimentation, coroutines, design patterns, perceived quality, software architecture

1. INTRODUCCIÓN

El concepto de arquitectura de software no fue estandarizado hasta hace unos pocos años atrás. La IEEE define el concepto de arquitectura como "la organización fundamental de un sistema dada por sus componentes, las relaciones entre ellos y el ambiente, y los principios que orientan su diseño y evolución" (IEEE, 2000).

Esta definición intenta capturar la esencia de varias interpretaciones que tiene el término. El Instituto de Ingeniería de Software (SEI, por sus siglas en inglés) de la Universidad Carniege Mellon es uno de los líderes en el tema el cual centra la arquitectura desde los puntos de vistas de estructural y más recientemente desde la vista de los procesos del ciclo de vida. Por otra parte, (Rumbaugh et al., 1999) establece la arquitectura como el resultado de una de las etapas de desarrollo en RUP (Rational Unified Process).

Independientemente de estas diferencias, existe consenso en cuanto a un subconjunto de elementos presentes en la mayoría de las escuelas. Estos elementos son los componentes, conectores y estilos arquitectónicos. Algunos autores incluyen otros elementos como los datos (Fielding, 2000), las vistas (Egyed, 2000) y también hay diferentes nomenclaturas paras las configuraciones, restricciones, y/o estilos arquitectónicos.

Varios autores (referirse a (Garlan and Shaw, 1993) para abundar) coinciden en que existen patrones arquitecturales o estilos que capturan la esencia de la organización un sistema o una parte de este y que permiten organizar el trabajo de desarrollo.

Algunos de estos autores sugieren que hay una relación entre los estilos arquitectónicos y los atributos de calidad percibida.

En este artículo se estudian dos implementaciones del estilo Modelo-Vista-Controlador (MVC) en sistemas Web que utilizan distintos tipos de conectores para la comunicación entre los componentes, y se analiza el impacto que tienen estas variantes en los atributos de tiempo de respuesta percibido, rendimiento y escalabilidad del sistema.

El resto del artículo se estructura como se describe a continuación:

- En la sección 2 se describe las variantes del patrón MVC, así como los elementos teóricos que distinguen estas variantes.
- En la sección 3 se establece el marco experimental que fue diseñado para comparar las dos variantes del patrón MVC.
- En la sección 4 se muestran los resultados experimentales y establecen las relaciones entre los resultados y los atributos de calidad.

2. MOTIVACIÓN Y TRABAJOS RELACIONADOS

Unos de los patrones más utilizados en muchos de los marcos de desarrollo Web es el Modelo-Vista-Controlador (MVC). Aunque el MVC se originó para resolver problemas de los sistemas con interfaces interactivas donde el usuario actúa sobre datos complejos (Reenskaug, 1979), ha sido "transportado" hacia los marcos de desarrollo Web de la manera siguiente:

- La vista por lo general se corresponde con una forma de representación de los datos en un estándar como (X)HTML que se interpreta por el navegador.

  Si bien es el navegador quien interpreta estos datos y dibuja los elementos en la pantalla, es también cierto que estos están condicionados por los elementos incluidos en el estándar. Así que se puede considerar que las etiquetas en las plantillas son instrucciones sobre qué controles el navegador debe mostrar. También se puede escribir CSS y javascript con propósitos de presentación.

- El controlador involucra tanto al navegador como al servidor. En cualquier caso el navegador es quien captura cualquier interacción del usuario y activa los elementos visuales correspondientes.

  En algunas ocasiones se puede invocar alguna acción al servidor. En este último caso, parte del controlador reside en el servidor, que recibe este "evento del usuario" y emite una respuesta apropiada. Esta respuesta tiene el efecto de modificar el modelo en el navegador, y por tanto la vista se actualiza en correspondencia.

- El modelo también está dado en los dos contextos: el navegador y el servidor. En el navegador el modelo se corresponde con aquella parte del DOM (Document Object Model) que es información a presentar[1]. En el servidor el modelo por lo general se corresponde con la estructuración de los datos que almacena el sistema.

1 De forma estricta todo el DOM es el modelo en el navegador. Se hace esta distinción porque se considera que aquellos elementos del DOM que son indicaciones de controles visuales se interpretan como sentencias de instanciación de elementos visuales y no como datos.

La relación entre estos contextos está dada porque el controlador del servidor transforma los datos a un estándar que sea procesable en el navegador. En última instancia, los datos enviados por el servidor afectan el DOM en el navegador.

MVC no impone ninguna restricción sobre la naturaleza de los conectores entre estos componentes. Solamente indica que:

- Los controladores se pueden comunicar con las vistas y los modelos.
- Las vistas obtienen datos directamente del modelo y pueden, en algunos casos, actualizar el modelo.
- El modelo puede notificar a las vistas sobre cambios.

En los sistemas Web, sin embargo, como los eventos del usuario ocurren en el navegador y las modificaciones deben propagarse al modelo en el servidor, no es común que la vista en el navegador modifique directamente al modelo en el servidor, y también puede hacerse poco factible que los cambios realizados al modelo en el servidor se actualicen en la vista de los usuarios[2], por esta razón el controlador en el servidor se convierte en un intermediario entre las acciones entre los dos contextos.

En la figura 1 se intenta capturar ambos contextos del patrón y las diferencias en los conectores. Se puede observar cómo el controlador en el servidor media entre el modelo y la vista, y el orden en que ocurren las interacciones en el servidor.

Aunque aparentemente el control del programa en el servidor está dado por ese orden algunas cuestiones merecen ser analizadas:

- En el estilo MVC no se indica que necesariamente estos conectores son del tipo llamada-respuesta. Además, como el caso de sistemas Web interactivos tiene algunas peculiaridades en relación al estilo original, se puede necesitar una revisión de los conectores para este caso.

2 En la actualidad se están popularizando tecnologías bajo el nombre genérico “Comet” que pueden aprovecharse para lograr este tipo de notificaciones.

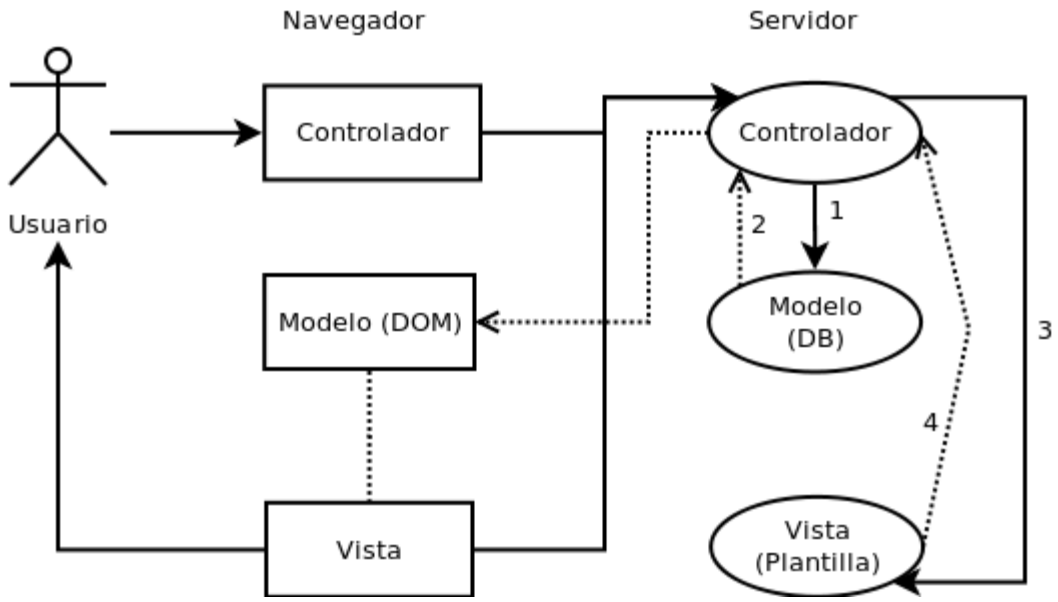

Figura 1: Representación del patrón MVC en ambos contextos de un sistema Web

- ¿Se puede emplear un modelo de productor-consumidor en este caso para los componentes del servidor? ¿Cómo se afectarían los atributos del sistema? En particular, ¿cómo se afecta el tiempo de respuesta percibido y la escalabilidad del sistema?

2.2. IMPLEMENTACIONES ACTUALES DEL PATRÓN MVC

Varios marcos de trabajo para el desarrollo de sistemas Web dicen implementar alguna variante del patrón demostrado en la figura 1. El autor tiene experiencia productiva con Django y el Zend Framework y ha revisado la documentación de otros.

Los conectores en estas implementaciones son invocaciones de métodos realizadas secuencialmente en el orden dado en la figura 1. Este modo de invocación, en la opinión del autor, afecta tanto el tiempo de respuesta percibido por el usuario, como la escalabilidad total del sistema. El razonamiento es el siguiente:

- Al recibir un pedido el controlador decide qué acción invocar sobre el modelo, la invoca y pierde el control. Si esta acción implica recuperar varios datos, el controlador no recupera el control del programa hasta que el modelo termina de extraer estos datos[3].
- Cuando el modelo recupera todos los datos requeridos retorna el control del programa al controlador. En este punto el usuario no ha recibido aún ninguna respuesta y todos los datos están almacenados en la memoria del servidor.
- Luego el controlador puede modificar de alguna manera los datos recuperados e invocar a la vista para obtener una representación de estos que pueda ser retornada al navegador.
- En este punto se pudiera liberar la memoria donde se almacenan los datos tal como fueron retornados por el modelo, pero puede que se necesite almacenar la representación de estos creada por la vista.
- Finalmente esta representación es enviada al navegador el cual puede presentarla al usuario.

3 En Django esto no es necesariamente así en muchos casos, dado que el modelo de datos en este sistema implementa una recuperación tardía de los datos: es decir, estos se recuperan en el momento justo en que se necesitan.

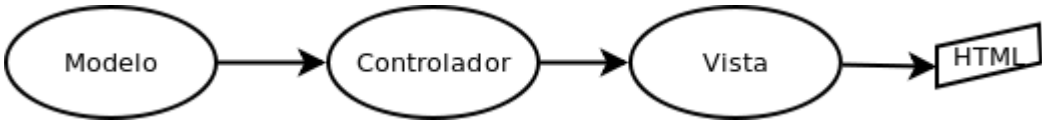

Figura 2: Estilo de tuberías y filtros para los componentes MVC del servidor.

Esta secuencia de acciones no cubre todos los casos posibles. En particular se asume que el modelo produce un resultado que debe mostrarse al usuario y que este resultado se puede procesar iterativamente. No obstante, este escenario tampoco es raro en sistemas empresariales donde el usuario necesita modificar y consultar muchos datos.

En opinión del autor una mejor opción sería usar una arquitectura de tuberías y filtros en el servidor. El modelo produce datos de forma secuencial y, mediante una tubería, el controlador los recibe, transforma y los cede a la vista; la cual transforma el dato a una representación apropiada y lo cede al navegador. La figura 2 representa este estilo.

En este esquema no es necesario que el modelo termine de recuperar todos los datos para retornarlos al controlador. Lo mismo se aplica a la conexión entre el controlador y la vista, y entre esta y el navegador. Por tanto, es de esperar que el usuario comience a recibir el resultado (parcial) de su operación en un tiempo menor comparado con el esquema de llamada-retorno.

Por otra parte, como los datos se emiten progresivamente al navegador no es necesario que el servidor mantenga todo el conjunto de datos en memoria. Se mantiene solo la parte que aún no ha sido enviada al navegador y ya ha sido producida por el modelo.

En la siguiente sección se describe un experimento realizado para probar si estas conjeturas se sostienen en la práctica.

3. MÉTODOS Y HERRAMIENTAS

Una forma de implementar el estilo de tuberías y filtros es mediante la utilización de corrutinas y continuaciones (Conway, 1963; Haynes et al., 1984; Wikipedia, 2009). Otros autores (Graunke et al., 2001; Queinnec, 2003) ya han explorado las ventajas de este tipo de mecanismo de control para la construcción de sistemas Web. No obstante, estos artículos se concentran en el poder expresivo de las continuaciones para capturar varias interacciones del usuario con el servidor.

Estos autores advierten que mantener un número de continuaciones activas por tiempo ilimitado puede comprometer la escalabilidad del sistema. Nuestra propuesta no mantiene ningún tipo de continuación (y por ende tampoco corrutinas) activas entre diferentes peticiones del usuario; por tanto se evita este problema.

Otra forma de implementar el estilo de tuberías y filtros es crear procesos concurrentes para los filtros y un mecanismo de paso de mensajes junto una cola para las tuberías. Sin embargo, este mecanismo sería mucho más complejo de implementar en el contexto de este experimento.

3.2. DESCRIPCIÓN DE LA APLICACIÓN WEB

Para conducir el experimento se construyeron dos servidores Web que simplemente retornaban una página HTML de 137,33KB con un listado de entradas en un directorio físico del servidor.

Uno de los servidores utiliza las corrutinas para implementar las tuberías entre el controlador y la vista. El otro servidor se implementó con la variante “clásica” de llamada-retorno.

El modelo está dado simplemente por una lista de entradas del sistema de ficheros. Cada entrada contiene el nombre, el tamaño, y la última fecha de modificación del fichero al cual representa. El controlador simplemente

obtiene el listado de entradas y los pasa a la vista junto con el camino del directorio del cual se obtiene el listado. La vista genera una tabla HTML donde cada fila se corresponde con una entrada del listado.

Ambos servidores se implementaron en Python y se utilizó el paquete Diesel para facilitar la construcción del servidor HTTP, debido a que Diesel utiliza las corrutinas de Python de forma nativa y se acerca más al modelo deseado.

3.3. DISEÑO DE LOS EXPERIMENTOS

Una vez que se tuvieron ambos servidores listos se corrieron doce sesiones de peticiones a cada uno de los servidores. Los parámetros de los experimentos fueron: (1) la cantidad de pedidos realizados, y (2) el tiempo promedio que se esperó entre el lanzamiento de un pedido y el siguiente.

Se utilizaron los valores 100, 200, 300 y 400 para la cantidad de pedidos realizados al servidor. Las ventanas de tiempo establecidas fueron 100ms, 150ms y 200ms.

Note que estos experimentos estresan considerablemente la capacidad de respuesta de servidor: Un servidor que recibe una petición cada 200ms, acumula 1.800.000 peticiones en una hora; si se acelera la velocidad a una petición cada 100ms se duplica este número a 3,6 millones. En el curso de estos experimentos cada uno de los servidores recibe un total de 3.000 peticiones, de las cuales un tercio se envían a la razón de 1u:100ms, otras 1.000 a 1u:150ms y el resto a 1u:200ms.

Por cada pedido se registra el tiempo (contado en milisegundos desde que se inició el experimento) en que se emitió al servidor, el tiempo en que se reciben los primeros bytes de respuesta, el tiempo en que se reciben los últimos bytes, y el número máximo de peticiones que se mantenían activas cuando ocurrió alguno de los registros anteriores.

Los experimentos fueron conducidos en una laptop con 1GB de RAM y procesador de 64 bits AMD Dual Core a 800GHz. El sistema operativo utilizado es el Ubuntu 9.10 para procesadores x86_64.

4. DISCUSIÓN DE LOS RESULTADOS

Tabla 1: Resumen de los resultados experimentales. Experimentos de 1u:100ms

	Clientes	100	200	300	400
MVC Clásico	Usuarios	28.18	95.94	104.13	115.64
	T. Percibido	3,870.03	8,232.78	11,654.82	15,567.36
	Duración	5,858.41	6,364.11	6,848.37	6,798.33
	T. Total	9,728.44	14,596.89	18,503.19	22,365.69
MVC Tuberías	Usuarios	27.53	86.96	104.83	111.36
	T. Percibido	3,181.62	8,135.77	11,471.33	13,849.09
	Duración	5,013.85	5,879.97	5,897.28	5,818.21
	T. Total	8,195.47	14,015.74	17,368.61	19,667.30

En la tabla 1 se muestra un resumen de los resultados obtenidos en los experimentos del conjunto de pedidos enviados a 1u:100ms. En el anexo A se muestran las tablas del resto de los experimentos.

Estos resultados ilustran que, de forma consistente, la implementación del MVC con tuberías se comporta mejor en tiempo de respuesta percibido y total que la implementación clásica.

Se nota también que la implementación con corrutinas acumula menos usuarios activos. Esto se debe a que como responde más rápidamente, las ventanas de tiempo (100ms, 150ms y 200ms) permiten al servidor liberar recursos de forma rápida. En el caso particular de la columna de 300 conexiones se nota que la implementación clásica

acumula menos usuarios, pero si comparamos la diferencia, vemos que es un caso aislado y que no se manifiesta de la misma manera en el resto de los indicadores.

Esta menor acumulación de usuarios es una indicación que el servidor implementado con tuberías puede responder a una mayor cantidad de usuarios que el servidor MVC clásico en una ventana limitada de tiempo.

Para decidir si las diferencias observadas en estos indicadores son significativas se utilizó la prueba de Mann-Whitney para comparar las sesiones de experimentos medida realizada contra cada servidor. Se utilizó todo el grupo de registros recolectados. Se utiliza esta prueba para ver si cada variable medida para cada servidor proviene de una población diferente, y también porque no se conoce la distribución de estas poblaciones.

Para calcular el nivel de significación sobre la diferencia entre los resultados, se aplicó el método de Monte Carlo con intervalos de confianza del 99%. Los resultados de la prueba dan valores entre 0 y 0,03, lo que demuestra que existen diferencias significativas en ambas muestras.

Esto quiere decir, en esencia, que la implementación del patrón MVC utilizando tuberías y filtros es generalmente mejor en cuanto al comportamiento de estas variables para los casos estudiados.

A pesar de obtener resultados positivos se debe observar que:

- La ganancia solo es notable en momentos de muy alto nivel de peticiones (más de 30 mil peticiones por minuto, lo que es equivalente a 1,8 millones de peticiones por hora). Además las diferencias absolutas no son mayores que unos pocos segundos.
- La implementación de tuberías no es factible en algunos lenguajes populares como el PHP.
- En los experimentos solo se logra la tubería entre el controlador y la vista, pero no entre aquel y el modelo.

Se necesitan de otros experimentos en contextos menos controlados para conocer la ganancia real y evaluar la factibilidad de crear marcos de desarrollo Web que incluyan este tipo de conectores.

Estas cuestiones deben evaluarse teniendo en cuenta la productividad del equipo. Se necesita de la creación de herramientas que permitan generar código eficiente, sin sacrificar la productividad. Ya en (Graunke et al., 2001) se demuestra la posibilidad de transformar un programa en otro semánticamente equivalente pero adecuado a ser corrido en un conjunto de iteraciones Web.

Aplicar este mismo esquema a lenguajes imperativos puede ser una tarea demasiado complicada, no obstante, se necesita investigar más profundamente para descartar esta posibilidad.

5. CONCLUSIONES Y TRABAJO FUTURO

En este artículo se muestra que una implementación del patrón Modelo-Vista-Controlador, en los cuales los conectores y componentes adopten el estilo de tuberías y filtros supera en términos de tiempo de respuesta y escalabilidad a la alternativa tradicional de llamada-retorno.

Para lograr el estilo de tuberías y filtros se usaron las corrutinas de Python, las cuales son mecanismos de control relativamente sencillos.

Se realizó un grupo de experimentos que apoyan esta conclusión para dos implementaciones del patrón en lenguaje Python. A los resultados de estos experimentos se les aplicó la prueba de Mann- Whitney para demostrar que el resultado en la implementación de corrutinas es consistentemente superior a la otra variante.

No obstante, PHP carece de corrutinas y no es elemental implementar este tipo de solución en este lenguaje. Por lo que se hace necesario continuar investigando en cómo ganar eficiencia sin afectar la productividad del desarrollo.

REFERENCIAS

Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. Tesis doctoral, The Royal Institute of Technology Stockholm, Sweden, December 2003.

Melvin E. Conway. Design of a separable transition-diagram compiler *Commun. ACM*, 6(7): 396-408, 1963. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/366663.366704.

Alexander Franz Egyed. *Heterogeneous view integration and its automation*. PhD thesis, University of Southern California, 2000.

Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Tesis doctoral, University of California, Irvine, 2000. URL http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Sitio web consultado el 2009-08-15.]

David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume I, pages 1-39, Pittsburgh, PA 15213-3890, January 1993. School of Computer Science, Carnegie Mellon University, Publishing Company, New Jersey. [Also appears as CMU Software Engineering Institute Technical Report CMU/SEI-94-TR-21, ESC-TR-94-21].

Paul Graunke, Robert Bruce Findler, Shriram Krishnamurthi, and Matthias Felleisen. Automatically restructuring programs for the web. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 211, Washington, DC, USA, 2001. IEEE Computer Society.

Christopher T. Haynes, Daniel P. Friedman, and Mitchell Wand. Continuations and coroutines. In *LFP '84: Proceedings of the 1984 ACM Symposium on LISP and functional programming*, pages 293-298, New York, NY, USA, 1984. ACM. ISBN 0-89791-142-3. doi: http://doi.acm.org/10.1145/800055.802046.

IEEE. Recommended practice for architectural description of software-intesive systems. Standard, IEEE, 2000. IEEE.Std.1471-2000.

Christian Queinnec. *Inverting back the inversion of control or, continuations versus page-centric programming*. SIGPLAN Not., 38(2): 57-64, 2003. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/772970.772977.

Trygve Reenskaug. *Thing-model-view-editor, an example from a planning system*. Note, Xerox PARC, May 1979. URL http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. [Consultado el 2009-12-01.]

James Rumbaugh, Grady Booch, and Ivar Jackobson. The Rational Unified Process. Addison- Wesley, 1999.

Wikipedia. Coroutine. Web page, Dec 2009. URL http://en.wikipedia.org/wiki/Coroutine. [Consultado el 2009-12-06.]

ANEXO A. RESULTADOS EXPERIMENTALES

En este anexo se dan los resultados de las tres sesiones de experimentos ejecutadas para comprobar la veracidad de la tesis planteada en este artículo. Se agrupan los resultados según de acuerdo a la ventana de tiempo entre el envío de un pedido y el siguiente.

Tabla 2: Resultados de los experimentos con 1u:150ms

	Clientes	100	200	300	400
MVC Clásico	**Usuarios**	28.07	41.49	47.42	76.55
	T. Percibido	1,293.57	2,991.83	3,340.41	7,000.31
	Duración	5,689.92	6,421.59	6,509.15	7,093.80
	T. Total	6,983.49	9,413.41	9,849.56	14,094.11
MVC	**Usuarios**	27.96	34.26	44.57	49.27

Tuberías	**T. Percibido**	1,090.36	1,764.48	3,302.10	4,214.11
	Duración	5,074.59	5,342.42	5,714.80	5,976.03
	T. Total	6,164.95	7,106.89	9,016.90	10,190.14

Tabla 3: Resultados de los experimientos con 1u:200ms

	Clientes	**100**	**200**	**300**	**400**
MVC Clásico	**Usuarios**	1.85	2.17	2.60	2.82
	T. Percibido	198.21	199.19	203.89	211.60
	Duración	254.05	244.69	334.22	398.73
	T. Total	452.26	443.88	538.11	610.33
MVC Tuberías	**Usuarios**	1.12	2.24	1.20	1.70
	T. Percibido	167.30	199.36	173.48	194.32
	Duración	65.68	265.45	76.90	192.25
	T. Total	232.98	464.81	250.38	386.57

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.