Property Context Model Framework

Pedro Dulce Serrano

February 17, 2016

Contents

Introduction and Concepts

1 Property Context Model paradigm

Property Context Modelling (henceforth PCM) is a paradigm of **software development design based on the application contexts properties** (*is similar to a model driven development approach*) and the relationships between properties of the different contexts.

The first time the Context Model appeared was in psychology studies published by Edward T. Hall, which I will describe in breve to point out the importance of context definition to understand an information system.

The development of applications based on PCM supposes a revolution in the manner of thinking and designing IT applications within the scope of Information Technologies.

For analysing the advantages of this design paradigm, I will start comparing it to the well-known Model View Controller paradigm (henceforth MVC), widely used by the Community of Object Oriented (OO) Software Development, and introduced by Trygve Reenskaug in 1979 in the OO Smalltalk programming language.

The MVC design fundamentals are characterized on the explicit distinction between the view domain and the model persistence domain, and the need of an interconnect element (*called Controller*) for data transporting between both in each Use Case.

Under the MVC design, the entities identified in an information system are often represented by classes with attributes, like Java Beans, as much in the View as in the Model Persistence. But under this modelling there are not defined classes for each attribute of each entity. This approach driven by entity at bottom-level design needs an individual definition within both domains for every property for all of the Use Cases where this entity appears.

Unlike MVC, the PCM design is focused on the similitude between the different information contexts, and for reaching this objective, at bottom-level the PCM design is driven by property definition, gaining a higher precision level than the MVC (which uses the definition of entity at bottom-level for modelling information).

PCM lies in these foundations:

1. The information of each context is defined once.
2. The value of information is unique, then, a data value is context independent.
3. Each element of information has a list of intrinsic attributes in each context.
4. The relationships between different contexts are defined once; the responsible element of manage the conversion and communication between contexts is the PCM controller.

In consequence, the definition of every property in each context gains achieves flexibility, and less code is necessary for applying the specific treatment of information in each Use Case.

Under the Property Context Modelling there are two identified contexts; context of view, context of strategy (*context with specific business rules*) and context of model persistence.

The PCM design solution can be applied as much for large client-server systems, as for small desktop applications.

2 Precedents of Model Context

Edward T. Hall pretends to explain how the context, the time managing and space affect to the inter-cultural communication.

Model Context points to the contextualization of communication: due to that humans receive multiple perceptive encourages and it is not possible to satisfy them with totality, the culture acts like a monitor that selects which stimulates are important for let them attention and how to interpret them.

Edward T. Hall, in base of context, defines two types of culture: High Context Cultures (HCC) and Low Context Cultures (LCC).

High Context Cultures (HCC)

High Context Cultures are charactered by the presence of context elements that help people for understanding the rules of language. The context is as much important as the words, social position (status) plays a decisive role in communication like the knowledge about that. In other words, the context appears like an important element in the semantic interpretation of the communication.

Low Context Cultures (LCC)

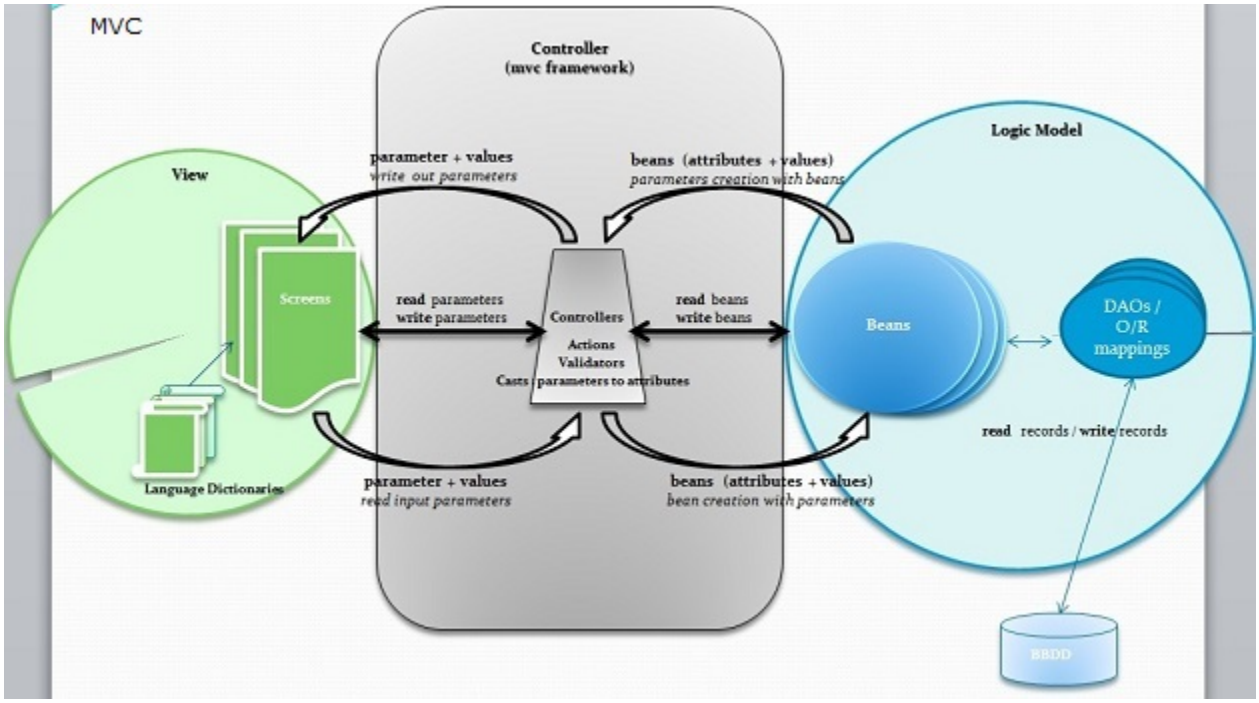
In the opposite, Low Context Cultures are charactered by a communication based on formal language; here the logical interpretation of information transmitted in a communication is very important. There is a separation level between people and the theme of the communication.

Saving the distances, we would see an equivalence between MVC paradigm and the LCCs because the context is not important for communication, and information needs to be redefined with precision for every step of transmission. Analogy, there is an equivalence between PCM and the HCC, where information is important, but it is not necessary to redefine in each transmission of information those elements with a lot of semantic charge that are intrinsically associated with the context.

These comparisons can help us to see the proximity between IT applications and knowledge systems: the importance of both yields in the definition of the particularities of each context that appears in a bidirectional communication, providing a conceptual and flexible model.

3 Comparing PCM with MVC

Model View Controller was presented in 1979 by Trygve Reenskaug in the context of Smalltalk, an Object Oriented (OO) language, and goes on been used (and not discussed) until today like a paradigm for the design and the development of IT applications.



Really, without entering in considerations about the validity of MVC, perhaps its triumph is the simplicity for describing the parts that compose a classic information system. I'm not going to discern about internal details of this design architecture, and I think is very useful specially for those situations where you need to transform data from input-to-output dispose, for example, for implementing a driver.

From the point of view of a montage string, there is an MVC scene when an operator receives the request of roll a torch (operator: Controller, result: View, torch: Model). Notice that the operator does not need to know the global target (contextual) that this action (control)has effected in the system.

Actually MVC yet is being used for design at high level complex IT systems, where there are business rules, entities and their relationships, user profiles,..., but I think is not the better way.

The problem I see MVC is the need for make and implement explicit validations, conversions, and mappings data between the user-interface (called view context in PCM) and the logic model.

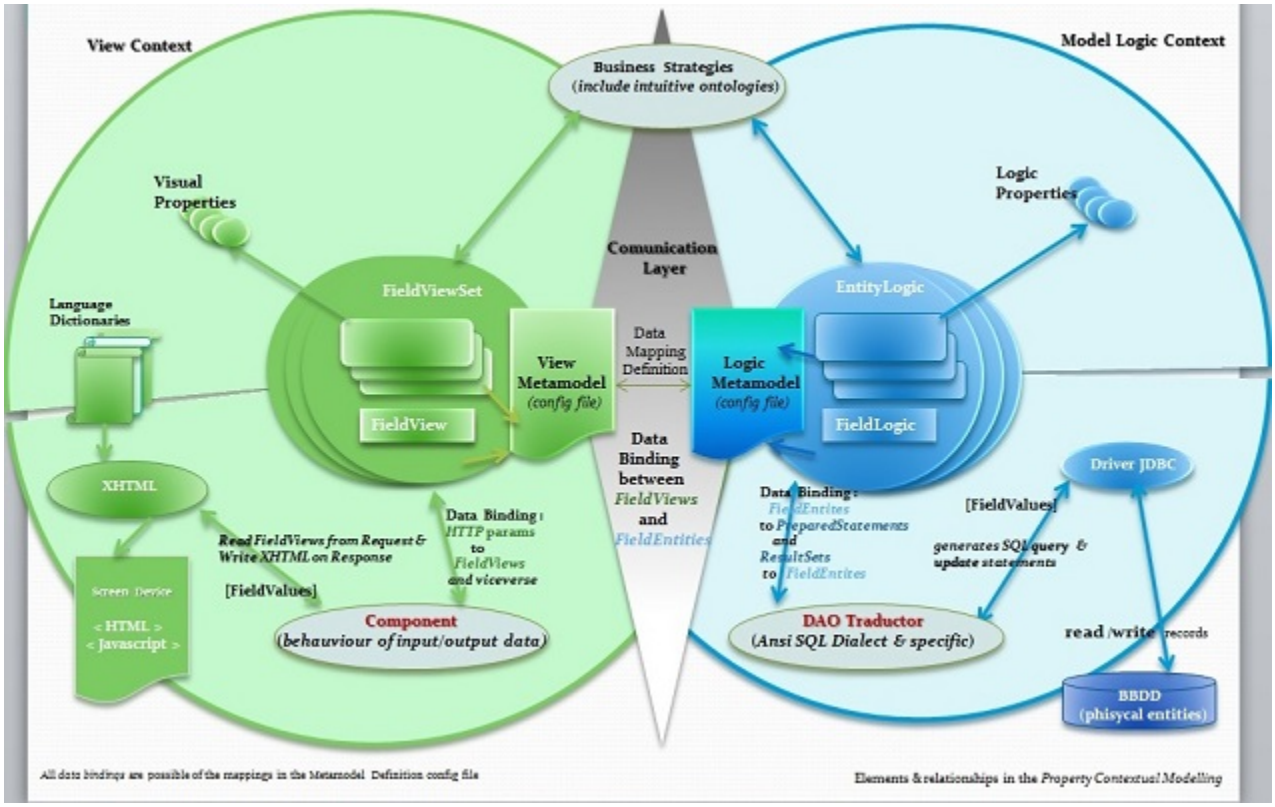
With PCM, developers don't need to focus their effort at making continuous and redundant code, because the framework is the responsible in all applications of applying that conversions between different contexts.

PCM opens a pragmatic and simple point of view for modelling these complex IT systems, with relationships between, not only the model and the view, else with business rules, user profiles, information mappings, etc.

There are two contexts of information data management of any IT system; the View Context and the Logic Model Context.

I am going to analyse the differences between both modelling IT architectures, depends of the tier:

- **Control tier**.



In MVC is necessary you make complex configurations to stablish the mappings between the View (*user interface*) and Data Model elements, deriving this in the need of implement controllers and specialized actions, for communicating both type of elements.

Under the contextual modelling, PCM, you define in the view metamodel the one-to-one relationship between elements of both contexts, View and Data Model context, so the generic controller already implements a cache for those application mappings. The control tier is a communication tier driven by the mappings written in the view metamodel; the control tier, when receives a request, creates and cache the specific components for that scene invoked.

The advantage is a lot of users are sharing memory for all the immutable information of the services defined in the application.

- **View tier**.

  In MVC, you usually need to develop the specific user-interface for all the scenes.

  In contextual modelling, there are a few templates predefined in the framework for creating the different user-interface patterns, for any scene. Each pattern is used by the framework view components at design-time, and in execution-time the framework creates a copy of the component to assure thread-safe for the data and the behaviour for each user request. The view component behaviour depends of view context ontologies, like the use of calendar when an element has date type, and uses the information of the elements defined in the view

metamodel file. The most common view components you need are already implemented by the framework, like a submit-form, pagination grids, header menu, navigation tree,... The advantage is developers don't need to implement .html or .jsp for each user-interface, because PCM framework paints dynamically them.

- **Parameter Binding**.

  In MVC, developers need to implement explicit validation and model-data conversion for each any of the string parameters that you receive in a HTTP-Request. This binding is necessary for all the scenes of all the application services defined.

  In the contextual model, developers don't need to implement those data-conversions, because the one-to-one relationship between a view metamodel element and an entity metamodel element is explicit for each scene, so the data-conversion is simple; this generic data-binding is implemented in an abstract component class of the framework. Another advantage in this tier for developers is that you don't have to name any element of the user-interface, because internally PCM framework assigns an unique identifier to an element (of the view component) when it is painted on the screen at the first time. Developers don't need to be cared about in how to make that conversions between both contexts, the PCM framework already give us that, in consequence, contextual model frees to developers of having binding bugs by mistakes with parameter names, or data conversions.

- **Actions tier**.

  In MVC you have to implement each action

  In PCM you don‘t need to implement generic actions, like query or insert, update and delete. Yo only define the operation of each scene in the view metamodel file.

- **Strategy tier**.

  MVC does not specify where and how the business rules have to be implemented in your the application.

  In the contextual modelling, there is an interface for this kind of classes which are the responsibility of manage the specific business strategies or business rules of the application. The concrete and specific value of any element of information in a PCM application is saved at the FieldValue class. This strategy tier has access, not only to the value of data, else to the properties of each element in both contexts, view and logic data model. The classes of this tier have to be implemented by developers; one default strategy that is already implemented in the framework is the authentication strategy class. With all the information in each context for an information element, and its value, in this tier you can take decisions depends of specific business ontologies (or rules).

- **Persistence Model tier**.

  In MVC, is necessary to use an O/R Mappings for make database access, or implement DAO operations with explicit SQL sentences.

In PCM you don‘t need to implement generic database operations, like query or insert, update and delete. Yo only define the operation of each scene in the view metamodel file. You have to be cared of define in the entities metamodel all the fields of each entity, and its data-type.

In the contextual modelling, under the entity model context, there is a DAO Traducer that implements SQL operations with the help of the entities metamodel file, the ontologies of this entities context.

4 Differences in context definitions with MVC

For understanding MVC design limitations, I will analyse its definition contexts comparing them with the definition contexts of the Contextual Modelling framework.

An information item in the Model tier in MVC is represented by a bean attribute, having a **name** (unique for that bean scope), a **type** and a **value**. An information item in the View tier in MVC is represented by a parameter, having a **name** and/or an **identifier** (unique for that user-interface scene), and a **value**. The Control tier is the responsible of making the validation or view parameters, binding them to the corresponding entity beans. Then Control tier invokes the appropriate action class, or invokes directly the DAO operation, passing those beans like arguments.

You can notice that in MVC the definition and the value of the information is not separated; both parts of information are saved in the same element, Bean.

Under MVC there is not an explicit definition for the intrinsic contextual properties of the information items. For example, if you want to paint a field like a html-input element always with a maximum of 8 digits, you need to write this attribute-value (*maxlength="8"*) in each user-interface scene where this field appears.

By this reason, you can not help you this explicit and external definition of data in each context for making generic and abstract algorithms for manage uniformly the data processing of an information system.

You can have an external and explicit definition of the Logic Model tier in MVC if you use an O/R mapping solution, like Hibernate, but I think those solutions are too complex and you need to be an expert in their use. And the disadvantage of its use is the same that is you use directly a DAO tier; you must define the queries and update persistence operations to use in each scene.

The advantage of the Property Context Modelling is the separation of explicit definition of context properties for information items, from their values or data.

The explicit definition of the information gets the benefit of making reusable code (algorithms) for the information treatment in the system. Because in PMC I distinguish two contexts of information, view and entities model, we need two explicit definition of the information, one for each context. Such explicit definition of the information items in each context, I call metamodel. I will explain in next sections the content of this metamodels. Is necessary that you know how properties you want to treat for each information context. In the framework released of PCM, you can find properties for the entities model context such as 'is primary key', 'type of data', 'maximum length', 'is not null', ... And for the view context, there are properties defined like 'maximum length', 'type of input', 'is read-only', 'is visible', 'is password',

etc. The number of properties of a context I call context dimension. So, you can notice that in PCM an information item has two property vectors, one for the view context properties, another for the entities context properties.

Understanding PCM

5 PCM context definitions

The element where you can define all of the services and scenes of your application, and groups all the relationships between the both contexts, the View and the Logic Model, is called **application metamodel**.

Analogy, the element used for defining specific entities, fields and properties of the Logic Model Context is the **entity metamodel**, and has a XML structure, designed from a relation-entity diagram.

The view metamodel has a XML structure too, where you can define profiles, components, services and action-scenes.

In the view metamodel, first, you need to declare the profiles identified in your IT system, and the static view components for all the scenes of the application, like the header menu or the navigation tree.

The most important elements in the application metamodel are the services and their actions. The service is the implementation of an Use Case, and you can associate it to one or more of all above defined profiles.

Each Use Case is decomposed in a set of Sub Cases, that are implemented by action scenes.

In an action scene you can set the different view components that compose the user interface, like html-forms and/or data-grids. Each view component of an action scene is equivalent to a sub-context.

Each view component (sub-context) has a set of sub-elements, that are called *fielviews*, and each fieldview has a mapping (direct relationship) with another element of the logic model, that are called *fieldlogics*, or to a virtual data or user-defined information. In last, each fieldlogic and fieldview has a set of context-dependent and intrinsic properties.

Examples of sub-contexts:

1. In the View Context, a sub-context groups all of the fieldviews and how to visualize them in a submit form,

2. In the Logic Model Context, a sub-context defines the intrinsic properties of the fields of an entity, like data-type, length, not null, ..., that represents a table in a relational database.

The dimension of an action scene has determined by the number of fieldviews defined.

There are two types of action scenes; transactional and query actions. - In **transactional actions**, you can plug a preconditions class, or/and a postconditions class. - In **query actions**, you can plug a preconditions class. Preconditions class are executed by the default PCM controller before the query or the transaction has been

initiated. Postconditions class are executed by the default PCM controller after a successful ended transaction. Both classes, pre. and post., are called ontologies.

Within preconditions and postconditions classes you can access to the different elements of each context, such as fieldviews of the View Context, such as fieldlogics of the Model Logic context, and modify their properties predefined in the application metamodel.

We call context dependent ontologies to all the (preconditions and postconditions) classes that apply rules for changing presentation properties of the elements of the View Context; for example, a class that change to read-only a selection combo field-view for a concrete profile user. A postcondition class that change the value of a field of an entity after a transaction represents a Business dependent ontology.

Here we show two examples of dependent context ontologies: - A fieldlogic defined in the Logic Model Context (in the entity metamodel) with its property 'is primary key' with value 'true' is always shown in a form entry of an update scene like a read-only html-input. - A fieldlogic defined in the Logic Model Context with its property 'data-type' with value 'date' is always shown in a form entry of an query, update or insert scenes like a html-input with a small calendar pop-up window.

In abstract, the content of the information shared by both of the contexts is defined in the application metamodel; how are those relationships between fields of both contexts is responsibility of the dependent context ontologies, and finally, why exist those relationships is responsibility of the Business dependent ontologies.

6 Controller layer in PCM

Under this modelling paradigm framework similarities between the both contexts are enforced.

The relationships between the properties of each context are the backbone of the interaction of an IT system developed with PCM.

In consequence, the controller layer only in PCM is unique for any application, and its unique responsibility consists of how to transport data from a context to another. In MVC, this controller layer is specific for each application; developers need to implement validations and transformations for any scene or Sub-Use Case.

In difference with MVC, under PCM developers does not need to define complex O/R Mappings between Logic Model Context and the database, or implement DAOs (data access objects).

You only need to define the fields, properties of two contexts **only once**; the entities metamodel, and the view metamodel, but they are not hurry about how those contexts are communicated. The how-to is responsibility of the PCM framework. Then, developers are hurried about what is the information in both contexts, and the mappings between them, not about in how to communicate or data conversions.

PCM framework applies this principle: You only need to dedicate your time to define once all the information between the both contexts in your application, and not how that information are going to be validated, persisted or transported through one layer to another.

The generic element implemented in PCM framework responsible of read and write information in the View Context is called **View Component**. And the element

implemented in this framework for reading and writing in the Logic Model Context is called **DAO Traducer**.

The indivisible element of information in the View Context is called FieldView; and the indivisible element of information in the Logic Model Context in called FieldLogic.

Each element, fieldview and fieldlogic, has a set of inherent properties defined in each context, in the View Context for the fieldview, in the Logic Model Context for the fieldlogic.

The Context Dimension is defined by the number of its inherent properties.

The set of fieldviews is called FieldViewSet, and the set of fieldlogics is called EntityLogic.

The value of and context property of a fieldview when appears in the View Context, or a fieldlogic when appears in the Logic Model Context, is unique and the same for the both contexts; a value does not depend on context. This value is called FieldValue in this framework.

For example, in general, student.age property has value '27' such as in the Logic Model Context, such as in the View Context.

7 View and Entities Metamodel in PCM

PCM manages two application metamodels.

A metamodel is represented like an .xml file, driven by an schema (.xsd).

In the entities metamodel you stablish how to group the database fields and their context-properties values.

In the view metamodel you stablish how to group the user-interface elements, their context-properties values, and their one-to-one relationships to the database fields.

The atomic elements of a view metamodel are FieldView. The group of fieldViews is called FieldViewSet.

The atomic elements of a entities metamodel are FieldLogic. The group of FieldLogics is called EntityLogic.

The relationship between one FieldView and one FieldLogic is defined in the view metamodel, analogy, the relationship between one FieldViewSet and one EntityLogic is defined too in the view metamodel.

8 Memory use efficiency comparison between MVC and PCM

In a web application, each HTTP-request usually transports the value of some data items. Under MVC, developers need to implement the treatment of each data item (validation and conversion) in base of its definition in the database.

In the contextual modelling, the references between the value of an item data and its definition under both contexts are in the two metamodels (*the view and entities metamodel*) so PCM framework can be responsible of treat validation and automatic conversion of each data item; its possible to affirm that treatment is not dependent on application. This explicit knowledge of the definition of the information let to

PCM to maintain in some caches al the information mappings of scenes, attribute definitions, view components,..., etc. In consequence, PCM makes an efficiency use of the memory of the server machine.

Memory used for managing the information and data for all of the user interfaces developed in an IT system under the point of view of both paradigms, MVC and PCM:

$$PCMmemoryused = \sum_{i=1}^{d} \sum_{j=1}^{a} k_{ij} \tag{1}$$

where

- d is the number of views defined in the application
- a is the pro-medium number of attributes in each view
- kij is the pro-medium number of k bytes of memory necessary for maintain the scope of the attribute j of the i view under PCM; for example, data-model-mappings, meta-data and JavaScript validation code, or its data-value

$$MVCmemoryused = U \sum_{i=1}^{v} \sum_{j=1}^{a} p_{ij} \tag{2}$$

where

- U is the pro-medium number of users accessing at same time to any view
- v is the pro-medium number of views visited at same time by users
- a is the pro-medium number of attributes in each view
- pij is the pro-medium number of k bytes of memory necessary for maintain the scope of an attribute j of the i view under MVC; graphic and behaviour characters, JavaScript validation code or its data-value

Assumptions: imaging an IT system with a high number of concurrent users, and a medium number of defined views. Also, we are going to suppose that the difference in k bytes between kij and pij are unappreciated, for this very common scenario, you can easily deduce:

$$U \sum_{i=1}^{v \cong d} \sum_{j=1}^{a} k_{ij} > \sum_{i=1}^{d} \sum_{j=1}^{a} k_{ij} \tag{3}$$

The clarity of the relationship on that formula is evident; PCM framework optimizes and does not penalize the use of memory in an IT system with high capability, with thousands of users, in opposite with the case of a classic IT system based on MVC. Also, such d and a are constants inherited of the IT system definition, when using PCM framework we can know a priori the maximum of memory used to serve with guaranties to the users connected to our IT system.

In the case of MVC, the use of memory can be exponential, in order that the variable U increases.

9 PCM Framework 01.00 released

Thanks to the abstraction capability of this contextual modelling framework, I developed a framework packaged in a pcm.jar file (≈400 KBytes).

In resume, these are some of the advantages of the contextual modelling framework:

- 1. you don't need to use O/R mappings for managing the data persistence and you don't need to make any query, or insert, delete or update SQL sentences.
- 2. You don't need to implement Servlet or another controller classes for classic maintain scenes.
- 3. You don't need to implement Bean classes for representing data entities.
- 4. You don't need to implement validation methods for each parameter.
- 5. You don't need to implement explicit type-casts for each parameter.
- 6. You don't need to implement explicit mapping from each parameter to any bean attribute.
- 7. You don't need to implement each user-interface scene.
- 8. Linear efficiency respect to the number of requests; this is possible thanks to the philosophy of caching definitions, view components, context element mappings, ..., defined in the application metamodels.

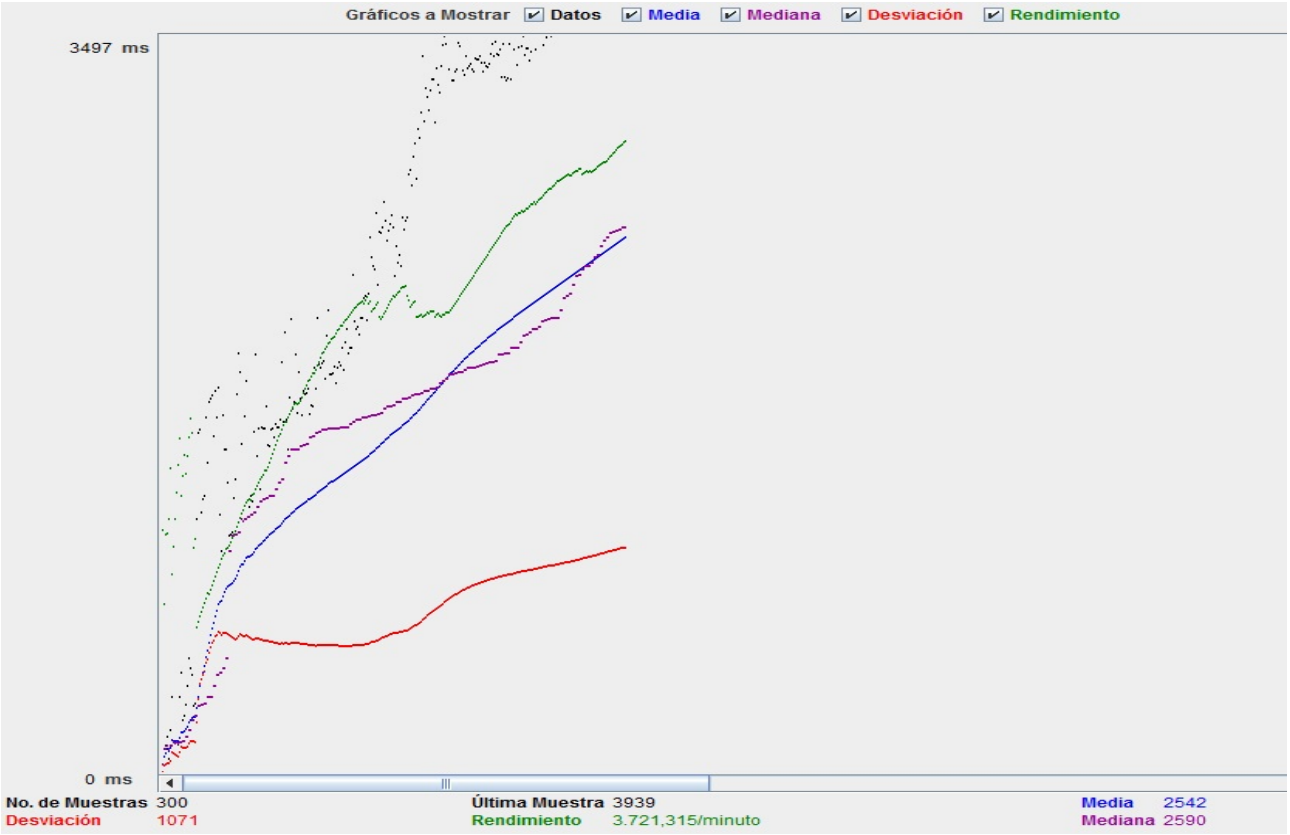
10 Framework efficiency

Here, I expose some tests results for showing the efficiency using PCM for a scene based on a paginated query.

Stress tests were passed to an application persistence managed by a SQLite database and running on JBOSS 5.0 server, in a 32 bits PC station with 4 GB of RAM and Intel Core DUO processors with 3 GHz.

- con 50 concurrent users: 57,5 requests/second
- con 150 concurrent users: 60,0 requests/second
- con 300 concurrent users: 62,0 requests/second
- con 400 concurrent users: 63,8 requests/second

A graphical view of the efficiency:


Gráficos a Mostrar
Datos
Media
Mediana
Desviación
Rendimiento
3497 ms
0 ms
No. de Muestras 300
Desviación 1071
Última Muestra 3939
Rendimiento 3.721,315/minuto
Media 2542
Mediana 2590

List of Figures

List of Tables