

The Moulage System

Core Description, Moulage version 1.0

The Moulage System is a real time data sharing & analysis system. It allows investigators to import data from virtually any source, perform real time analysis of the data in question, query other available data sources (networked database systems) and export the data to a colleague which includes the analysis software needed to view the data, or to a formatted report, binary data file. The Moulage System can also provide access to high performance data systems to allow large computational analysis of data at speeds normally inaccessible to the investigators.

The Moulage System has been built exclusively in the Java Language from JavaSoft a subsidiary of Sun Microsystems. The Java language is a compile once run anywhere programming solution that has been embraced by all major and most minor computer OS manufacturers. By constructing the entire system in Java the Comprehensive Biology Project has provided a completely transparent distribution capability of all related software.

The Moulage System is fully multi-threaded for parallel or multi-process operations. The design has taken into consideration the ability to perform multiple analysis, data loading and data mining operations simultaneously. In order to perform multiple tasks simultaneously the entire system must be capable of preventing single resource corruption by multiple processes attempting to update that resource at the same instant. The design enables the Moulage System to take full advantage of modern multi-process platforms and parallel operating systems.

I. Design Goals

The Moulage system was designed with the following goals:

A. *Transparent access*

Allow users to use the extensive services without being concerned with the origins of the software or data. All needed data or analysis methods are retrieved dynamically including from remote hosts on demand.

B. *Accept multiple data sources*

Allow importing and exporting from any source data no matter what format it is in.

C. *Allow easy sharing of data*

Create an environment that allows email, news groups and other communication technologies as well as data mining to share the data with concerned colleagues.

D. *Keep data in form usable by object-oriented databases*

Due to the nature of most items being analyzed an object based approach is needed to allow for the diverse types of data. Since the data needs to be in an object form to analyze storing it in that form is the best approach both for speed of retrieval and data validation.

E. *Provide multiple tools for analysis*

Various data types can be associated within many different analysis. Thus provide a means to define analysis tools to be cross compatible.

F. Dynamic version distribution

Allow the users to always be assured they have the most current version of the software without constant checks of a site combined with installations.

II. Base Components

The Moulage System has been designed to be fully modular. Each component performs a small subset of functions that are specific to that component only. Most of the inter-object interfaces and functions have been built into each component at the Moulage class level.

The design allows for easy implementation of new classes to the system by any collaborating PI. Creating a new Inspector (analysis interface) that reuses other data types is built simply by extending (inheriting) the Moulage Inspector class and adding in only the new analysis methods. Loading of the source data, network communications, database interactions and universal user interface components (i.e. open new file, added data set, view textual source data) are already written and functional immediately. Creating an entirely new analysis environment including new data types is done just as easily with the exception of a new Imprint (the object representation of a specific data type) must also be defined.

Moulage System components can be enhanced or replaced at the Moulage class level without effecting current systems. By defining the application programming interface [API] all of the underlying methods can be enhanced without introducing problems to collaborating PIs. By the use of dynamic class loading across the InterNet these enhancements would immediately be integrated into all sub-classes transparently to PIs.

The entire system has been in an object oriented environment allowing for complex data relationships to be modeled. The next sections describe the various Moulage System components. The following diagrams are provided to enhance the descriptions. Figure II.C.2-1 shows the Moulage System interface components API layer diagram depicting basic layering of the analysis environment. Figure II.C.2-2 shows a network and API layer diagram of the Moulage System. Figure II.C.2-3 shows the system configuration and general object interactions during a data acquisition cycle. Finally figure II.C.2-4 shows the base object interactions during an interactive analysis after all of the data acquisitions have completed.

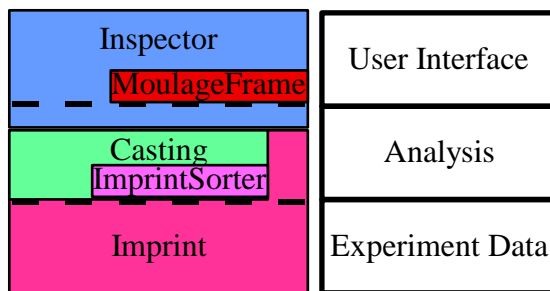


Figure II.C.2-1 Moulage Component Layer Diagram

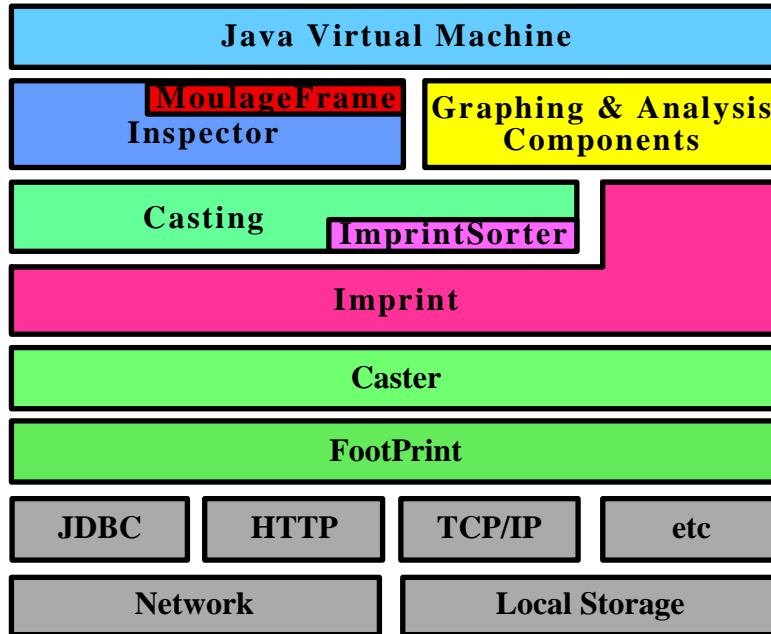


Figure II.C.2-2 Network & API Layer Diagram

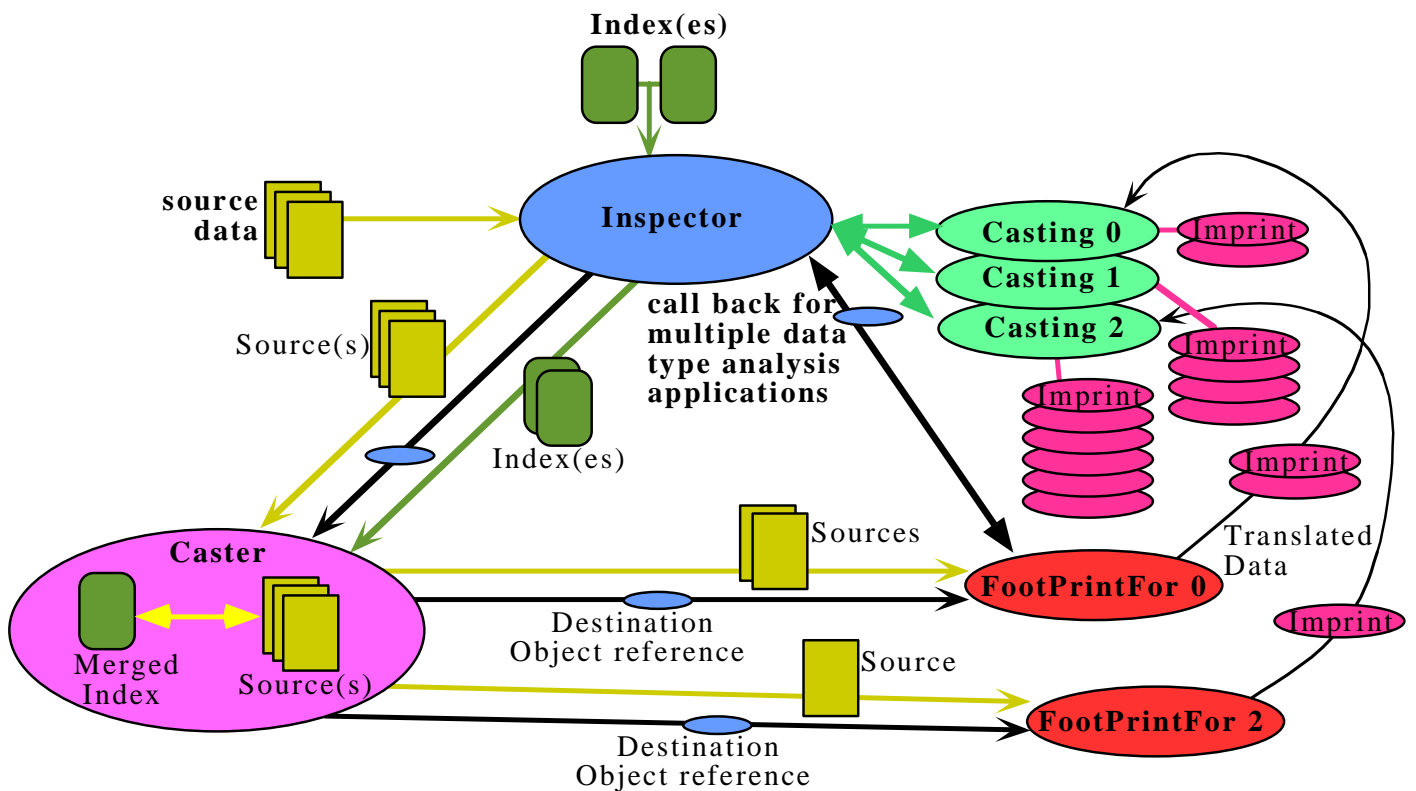


Figure II.C.2-3 Data Acquisition

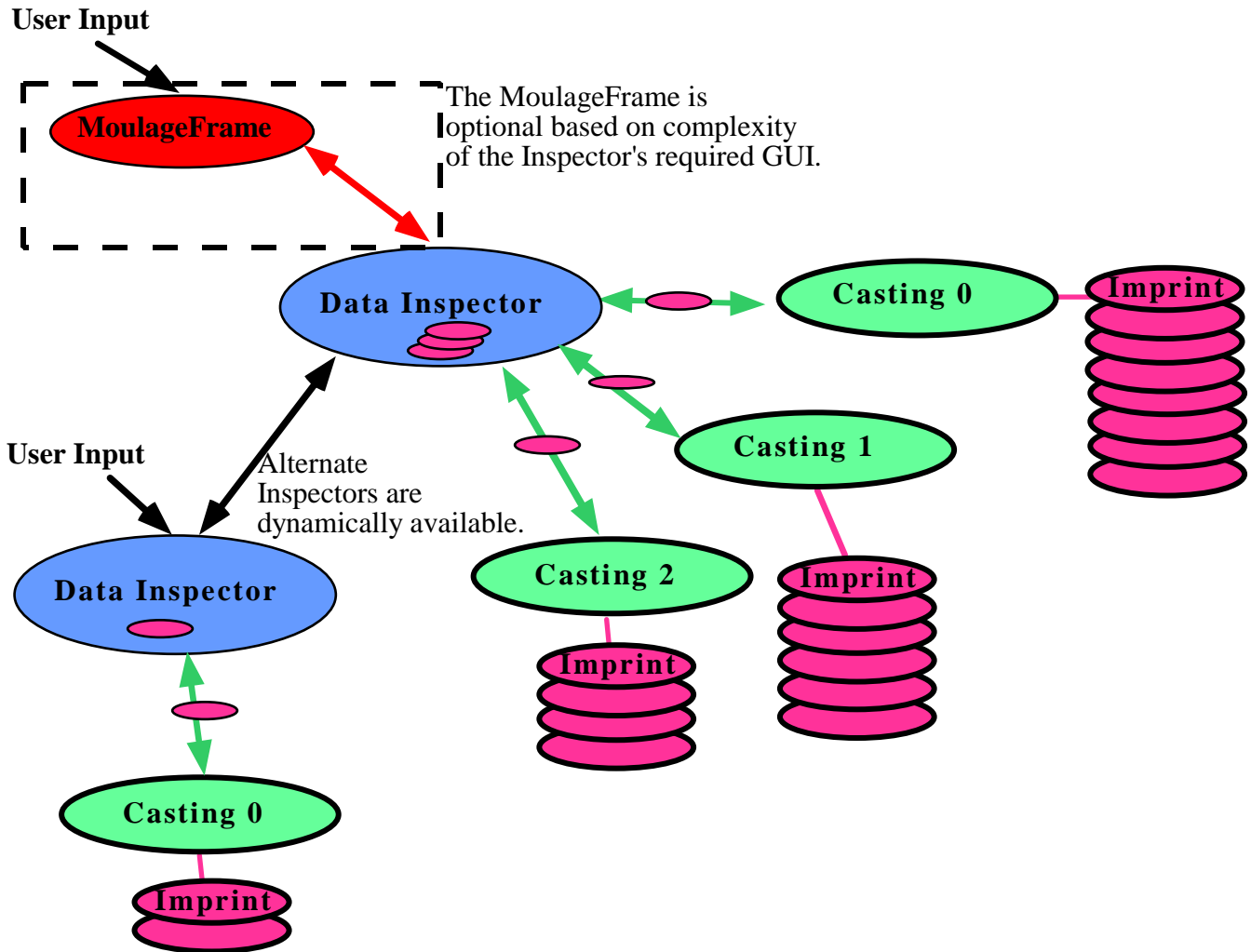


Figure II.C.2-4 Interface Relationship Post Acquisition

A. Inspectors

An Inspector is an analysis tool that performs a specific set of analysis functions against one or more sets of object data types either autonomously or in real time providing a visual graphic user interface [GUI]. Inspectors are the keystone class for all of the Moulage System providing for analysis of data via a large number of execution methods. The base Moulage Inspector class provides for most generic operation controls and interfaces it is designed to assist with extending its uses for specific analyses.

The Inspector API methods tie back to the Moulage System's primary design goal of dynamically linking all necessary routines for any type data. Thus, allowing for a completely heterogeneous environment that is well suited to analyzing multiple data types in a single intermixing visualizer to provide for collaborating, exposing, and database interfacing.

Inspectors provide for execution via a command line interface, icon windows launch, an applet through a network browser and via interface calls from other Inspectors. Input parameters are generally defined so as to maintain a common interface logic. For example, when called through a web server as an applet the logic for setting up the internal references like Debug, FileBase, etc. are already handled within the base class.

If an Inspector is launched via a browser it automatically looks for the applet parameters:

1. "Debug" which can be any of "ON", "OFF", "true", or "false", not present means OFF
2. "Filter Index" Which is a white space separated list of the index files to use for this instance. These can be URLs, direct file references, or pointers from the current FileBase.
3. "Input Files" Which is a white space separated list of the source data files to load. Again these can be URLs, direct file references, or pointers from the current FileBase.

All Inspectors have a cross Mouflage interface for being called by other Mouflage Inspectors. This interface or object method is meant for starting up the data object analysis by passing object references between the Inspectors. This allows for Inspectors to be tied together or reference other analysis tools that work on the same types of data. Mouflage components built by the CBP have also established a precedence for using this same method for all startup methods. Hence there is only one startup route no matter the execution method. This allows the Inspectors to easily deal with multiple directions of start up (i.e. from another Inspector, as an applet and from the command line).

Another API method built-in to the Inspector allows for cross analysis communication. This method is called when a Mouflage component is looking for some piece of data available from the Inspector. At the Mouflage level the function returns the items title, FileBase and nutshell by default. The method is required to pass back an appropriate instance of a new object when called with a name or package id of the requested data type or null if unknown.

B. Caster

The Caster a dynamic source identification and import class. This class reads source files, index files (of keys to FootPrints) and a data object to hand off to the FootPrint(s). It analysis all of the sources to determine their type by using an internal set of logic and against index data. Once a source has been identified it is either: converted internally and inserted into the mini-real-time OODBMS for analysis; or the caster dynamically loads a FootPrint to perform the data translation from the source data into the mini-real-time OODBMS for analysis.

The Caster is designed to be a completely transparent program outside of the users perspective. It performs data loading, source analysis, dynamic class locating and other functions completely independent of other operations underway.

C. FootPrint

A FootPrint is a small independent data filter responsible translating from one specific data input format to a specific object representation of that general data type and vice versa. The base Mouflage FootPrint class contains the normal interface controls for communications with the other Mouflage classes. All of the data source handling, type checking, reference control, etc. is handled automatically.

A FootPrint is needed to perform the import and export functions for each data type. Collaborating PIs that wish to use an Inspector on their unique or proprietary data need merely to create a new FootPrint on their host to import their data. FootPrints can be public or private. Hence internal data types can be mixed with external data without compromising proprietary files or data.

Creation of a new FootPrint is achieved by extending the base Mouflage FootPrint and writing at least three simple methods: a short initialization method to describe the FootPrint to the executing Mouflage components; an import method which only needs to translate from an already loaded file to an already type checked Casting instance; and an export method to translate from an Imprint to an output byte stream for exporting the data back into the original data file format.

The initialization method is a very small method called "andMore." This method is a constructor method for setting up a few needed variables: "version" is recommended to be modified by placing a class name, version data, and copyright statement onto the front end of this string; "thisClass" is a full class name identifying this extension's name which is used for cross references with other components and for debugging output; and "footPrintFor" is required to be defined to a string which contains the full name of the Imprint this extension is designed to write too.

The import method is called "makeCast." It must have the code to read data from a passed in Java ByteArrayInputStream (AKA the source file) and translate it to the appropriate Mouflage Imprint. Plus this method must also make sure to add the translated Imprint(s) into a Mouflage Casting instance which will be passed to makeCast() by reference.

The export method is called “imprintCast.” It needs to process a passed in array of Imprints into one or more output files formatted to be acceptable source data. This output files will placed into an Java ByteArrayOutputStream(s) in an array returned to the calling method. This method allows exporting of any Moulage data to a local format for use in a local program external to the Moulage System.

The base Moulage FootPrint has the methods to receive data source file(s) as: a preloaded ByteArrayInputStream, in which case there is only ONE file to process; a string of file names (URLs, direct path or some appropriate combination, a lot of effort is spent trying different means to read in the source files) as space separated names; or an Object reference which will be checked against the expected Moulage Casting type (defined in the footprintFor string) and added into the appropriate Casting instance if it is appropriate.

The FootPrint class is designed to receive the destination reference if only one type of data is being loaded. However, the reference first passed to the FootPrint should normally be a Moulage Inspector type (this is also checked). If it is an Inspector a call to the back channel method (“needPlaster”) is initiated to ask for the appropriate Casting type. Once the first call for a Casting reference is performed the reference is typed checked again. If is bad (null) or of the wrong type at this point the FootPrint fails out with a call to debugging logging (“tattle”) and exists nicely returning control back to the calling object.

D. Casting

A Casting is a generic, independent object manager for Imprints and their related ImprintSorters. It provides a complete OODBMS style API used by the other Moulage components.

For each data type to be analyzed an Inspector will create and initialize a Casting instance. Once a Casting has been given an Imprint it adjusts its internal methods to handle only other Imprints of the same type. A Casting instance can be recycled by clearing it of all Imprints and handing it a new Imprint of a different (or the same) type.

All calls for sorting, enumeration, and direct random access to the data elements are handle internal to this class. All returned data is either stored in the external public references for this class (as defined for each specific data type) or placed into an external Imprint object instantiated for just that purpose.

Currently only one element can be called for from an instance of a Casting at a given time.

Castings are set up to be thread safe. They treat all internal data as immutable. If an object is retrieved from a Casting what is actually passed to the Inspector is a instance clone of the internal Imprint. The internal element can be changed by overwriting it with another Imprint. But, that is a deliberate action. This also provides an automatic “undo” type function to the Inspectors since they are only working with copies of the data.

Although the Casting class is thread safe there still exists the possibility of an input problem if external objects of type Imprint are being used by an Inspector analysis code which is also being added into a Casting while it is being changed. All methods must be careful with which object it is changing. This class does gain a synchronized lock on external Imprints when extracting or adding. But, there is still a race condition before that lock is achieved for adding. Data extractions are not affected. However, Moulage has a built in safety mechanism for validating all input data objects within each Imprint that will at least detect “improper” data.

E. Imprint

An Imprint is a specifically designed data type object (e.g. a ‘protein’, a ‘DNA sequence’, etc.). This class is individually designed for each type of data it represents and holds data relevant to a single instance of its type (e.g. 1 DNA sequence, 1 protein, 1 gene, etc.).

Each Imprint has two special methods. These are validation methods for confirming that the Imprint holds reasonably good data. Both of these methods must return a boolean (true or false) that indicates if the instance is a valid data representation of the type of data the Imprint was designed to reference. While performing these checks use the methods are expected to use calls to the debugging method “tattle” to describe the check progress (especially a failed check with the reason for the failure). This allows for validation and return messages to be passed to the user or data management system that can later be used for cross checking or debugging purposes.

The first of these methods is called "perstringe." The perstringe method needs to perform any and all quick surface level (high speed) checks that can be performed in real time without adversely affecting run time. This method is called by an instance of Casting before it is added into the internal list of Imprints.

The second method is called "battue." The battue method needs to perform any and all deep data mining validity checks that can be performed without concern for run time. The battue method is meant to only be called from inside of the large OODBMS repository. This method by definition can take some considerable time to complete. This is the primary means by which the automated experiment revalidation will be implemented within the CBP's OODBMS system.

F. ImprintSorter

An ImprintSorter provides the correct methods for comparing one object to another based on some type of method(s). Additionally this method is to be used for locating a specific data object within a Casting. This requires a lot of thought for each individual Imprint or data type.

An ImprintSorter must be named after the Imprint it was made for. The name must be package of Imprint + Imprint class name + "Sorter". Nothing else will work. This standard is used so that a specific sorter can be located and loaded at set up time for a Casting via a call to the Java classloader in the form of `Class.forName(Imprint.thisClass + "Sorter")`.

The bases for the ImprintSorter compare method is that for any type of data there must be a means to order it for visualization and reporting purposes. For data types that this is not valid for no ImprintSorter need be defined. It is not an error if there is no associated ImprintSorter. All comparisons are of two elements at position A and B: true should be returned if item A is less than (<) B; a false should be returned if A is equal to or greater than (>=) B. All sorts are on the bases of ordering the lists from smallest to largest quantities. You may reverse that method by changing the return values here. In any case if A is equal to (=) B a false should always be returned to prevent un-needed swapping of equal elements.

The considerations for the compare logic are:

1. The first Element is always considered to be closer to the 'start' of the sorted list of data elements.
2. if the two elements compare to be the 'same' value return a false.
3. Return a false if the compare can not be completed.

III. System Operation

All entrances to a Moulage System analysis are through an Inspector (the analysis interface for both real-time GUI interaction and automated analysis routines.) The Inspector is passed none, one or more optional inputs. The initial input is normally a list of source files, a list of index files to use for the data load, and debugging status. If the Inspector is running as a real time user interface these items are dynamically modifiable by the user during execution.

Each data source is passed through the Caster, which identifies the source type and associated FootPrint class (the translator or data filter.) The Caster then attempts many approaches both local and network accesses to dynamically load the needed FootPrint. If it fails it reports the failure and moves on. If it succeeds it constructs the FootPrint instance setting thread priorities and debugging state. Then the Caster passes the new FootPrint instance the source data and instance references. The FootPrint breaks of into its own thread and begins loading the data. The FootPrint(s) filter the data source(s) into the needed Imprint Class(es) and add the new object(s) into the specified Casting object(s). Once the FootPrint is running the Caster then begins work on the next source data. This continues until all sources have been passed to an appropriate FootPrint or were unable to be loaded. The Caster then waits for all FootPrints to finish and it calls back to the Inspector notifying it that the load is complete.

The Caster uses an internal logic to control the number of FootPrints operating at any given instant to keep from overrunning the CPU of the system. The Caster can run single threaded or fully dynamic.

The Inspector does not need to wait for the Caster to finish before moving on to other tasks. In fact if a user wishes to add in other data sets via the GUI additional Casters are created to handle the additional data.

As each Imprint is added into the Casting instance(s) the data is spot checked by the Casting before adding the data into its list. One design criteria for the Moulage data structure is to provide high data integrity. Data integrity is defined to mean, in part, that any data within the system has not accidentally or maliciously been altered. To ensure data integrity the Imprints should be designed to assist in determining if data has been altered to 'look' valid without being valid. Data integrity also means that only the individuals who have been explicitly given authority over the data objects can add, alter, or remove the stored information. In building the Moulage System scheme the CBP has provided a data object class that can not be altered except through its interfaces. This controlled interaction with a Moulage Casting based data object allows us to build into the object any and all integrity and security checks that are needed to provide a high level of trust in the data itself.

The Moulage System is designed to provide a scaleable approach to ensuring data integrity. As more data, of a specific data type, is entered, internal, user independent criteria will be established to compare elements for consistency. The criteria are derived therefore from an ability to parse an element of a particular data entry and compare it against itself as represented in some other data structure. For example, within the Imprint for mass spectrum data the sequence of the peptide resides within a file containing the sequence of the protein in which the peptide region is represented. As each entry to the Visualize Mass Spectrum Inspector occurs, a separate method within the Moulage Imprint is called to check the peptide sequence with the protein sequence file and confirm a match. If the match is confirmed, the object is loaded without alteration; if the match does not occur, an alert [a visual indicator '*'] is inserted at the beginning of the sequence entry. Additionally when data is inserted into our data archives the archive data structures will perform intensive data checks to validate all entries. For all Inspectors the source data passes through FootPrint and is added into one or more Moulage Casting(s) during initialization. This includes data already in a Moulage Casting format coming from a larger archive or database. Hence, every time the data is loaded the Moulage System performs these checks.

After all data sources have been parsed into appropriate Moulage Imprints and inserted into one or more Castings, the Castings become mini OODBMSs. These small, self-maintaining databases allow various Inspectors, data analysis tools also dynamically available across the InterNet or intranets, the capability of providing real time visualization, document composition, data analysis, information exchange, or/and database insertion.

IV. Functional Inspectors and Under-development Inspectors

This section provides short descriptions and some details on the currently functional or in-development Inspectors. It must be emphasized that this list is considered incomplete as the Moulage System may be extended and used by any lab with access to the system classes without the CBP's knowledge as we freely provide access to the APIs that instruct other programmers how to extend the various Moulage classes. All extensions of the Moulage System have access to all of the globally available features without breaching the security of any collaborator's experiment data. Additionally as the system is expanded any extension that references our site's classes will receive all the new features and functions without ever having to modify or recompile their extensions. This system is entirely reusable and backwards compatible.

A. Data Manager

The Data Manager Inspector is a Moulage System Component that was added to the system to manage other Inspectors. It accepts data from source files that describe available data sources, related index file sources, related Inspectors, a short title and a long description. Its provides a GUI listing of available data and the associated Inspectors in a central control panel. It also communicates with currently running Moulage components and can control their operations. All Moulage process operation components (Inspectors, FootPrints, Caster) report to this Inspector if it is presently operating. The individual processes are displayed on a list window panel for the user to observe the systems activities.

Currently this Inspector is used to provide a web resource GUI listing of all available data within a collaborators web site. The CBP is using this Inspector to separate collaborating labs data sources into independent locations or sub-domain web pages.

The Data Manger is capable of shutting down one, some or all currently running Inspectors from its control panel. The ability to manipulate Moulage Components thread priorities has just been added into Moulage as of version 2.1. This Inspector is slated to allow user manipulation of these priority threads from the control panel to allow for dynamic reassigning of process priorities by the users.

B. Thread Snooper

The Thread Snooper Inspector is a unique implementation of an Inspector. It does not accept data from a remote source. Instead it uses the current Java Virtual Machine as the source of its data. This Inspector is actually a debugging tool that displays all currently running Java process threads. This was constructed as a Moulage Inspector as a demonstration of the generic application abilities of the main system design. It is being enhanced to also allow for thread priority manipulation under the Moulage version 2.1 enhancements.

C. Applet Window Control

The Applet Window Control Inspector was designed to allow for easy access to any applet. All Java applets are specified to be accessed and started in a certain set method. This Inspector is currently being designed to be an integral Moulage component that will allow any applet to be run within the Moulage System. It accepts one or more applets which are then started in independent windows with all of the expected applet interfaces in place.

Additionally this Inspector is intended to eventually allow for easy construction of other Inspectors. A programmer can target the individual Inspectors for only an applet environment and then use the Applet Window Control Inspector to provide for independent execution via the command line or Data Manger.

D. Consensus

The Consensus Inspector is being constructed by a collaborating lab at the University of Colorado under the Direction of Dr. Gary Stormo. This Inspector is intended to provide the Moulage System with direct access to the consensus sequence analysis system built on Dr. Stormo's research.

The consensus application performs complex analysis on one or more sequences (DNA or protein) to locate unspecified pattern similarities within the input sequences. This system provides a large number of analysis options and can perform analysis on database retrieved sequences.

By converting the entire application to Java and integrating it into a Moulage Inspector Dr. Stormo intends to allow for easy access to the methods as well as allowing it to be tied into alternate Inspectors which have the ability to work with sequence data.

E. Detente

The Detente Inspector was built by the CBP during the early stages of the Moulage System construction. This Inspector manages external data sources, allowed query construction and response controls. Each data source, by definition, will have one associated FootPrint, this includes data base systems. The CBP is constructing FootPrints to query external data sources like Gene Bank and Swiss Prot. The Detente Inspector will provide access to these FootPrints directly or may be used by other Inspectors to perform these external queries. This Inspector also allows for controlled analysis of all returned data.

The Inspector provides a GUI to construct 'allow query' matrixes (associate genes, protein matches, etc.). Once a query has been built it can be targeted at one or more data resources and will be appropriately converted for each source. The Inspector then starts the queries and waits for the return data. The returned data can then be directed into one or more Inspectors for automatic correlation or analysis. Once analysis is complete the results may be used for additional queries and finally sent to the user requesting the queries via email, a data base insertion, or, if the user is waiting, directly starting the appropriate Inspector with the new data.

This Inspector has been constructed. However, the complete FootPrints for external data source querying have yet to be completed. The CBP intends to construct these FootPrints as collaborating labs' needs dictate. As the FootPrints are developed this Inspector is expected to be considerably enhanced to allow very fine control over data mining.

F. IONgraph

The IONgraph Inspector is built of a number of independent modules that perform spectrum display, manipulation, and sequence comparison to spectral data. This Inspector allows for a dynamic manipulation of mass spectrum data.

The Inspector is capable of dealing with input spectral data. Which is then displayed within a GUI. The GUI allows for manipulations of the spectral data such as (this is not a comprehensive list): background filtration; normalization to X scale; precursor extraction; top N peak extraction; and peak smoothing functions. Additionally the Inspector accepts a sequence that the spectrum may be representing and calculates the theoretical spectrum of the sequence and shows correlation data between the displayed spectrum and the theoretical spectrum (after it has been manipulated similarly to the displayed spectrum). The sequence calculations also allow for the user to edit or input alternate sequences and performs the same calculations on user supplied sequence data.

The IONgraph Inspector was constructed to add additional functions for analysis of mass spectrum data under the NIH Resource Center. It is currently being tested by one of the NIH collaborators. It has proven to be very efficient and useful. The CBP expects to be releasing this Inspector to all of the collaborators by 1st quarter 1997.

G. *Sammengi*

The Sammengi Inspector extends early NIH funded research and development of the quantitative analysis and comparison of 2D gel images of proteins [Garrels, 1988; Garrels and Franza, 1988a & b]. The Sammengi Inspector is a re-engineering of a 2d gel analysis package application. It uses multiple gel analysis tools as examples and is an entirely new product. It, however, uses previous gel analysis methods available from the Quest II Project and from the Gal Tools gel analysis applications.

The Sammengi Inspector accepts images of 2D gels, or other shaped densities like the 'bands' in a 1D protein gel image or DNA sequence image or "spots" from a DNA array image. These images can then be selectively analyzed via one or more automated methods in full three dimensions (length, breadth and depth of grayness, or OD). The Sammengi Inspector imports and analyzes any images captured from a wide variety of imaging devices provided a FootPrint is available for that devices output data type. Encoding such FootPrint class extensions is straightforward and described above. The Inspector allows the user to fine tune the analysis through GUI based editing and input adjustment features. The peaks may be adjusted manually or automatically for use on awkward gels with spots of unusual profiles (e.g. the spots of some gels accept silver stain in an uneven manner). In addition to spot detection virtually all types of lanes can be analyzed including horizontal, vertical, overlapping and variable sizes.

The Sammengi Inspector currently being constructed by the NIH Resource Center. It is projected to be available by 4th quarter 1997.

H. *Visualize Mass Spectrum*

The Visualize Mass Spectrum Inspector allows for a method for performing protein identification & peptide sequencing by utilizing analysis routines that take mass spectrometry fragmentation patterns and search protein and nucleotide databases. This Inspector then provides for analysis of these specialized applications reports in a single comprehensive application.

The CBP used the SEQUEST application developed by a collaborating lab (the John Yates lab) as the first application to interface to. After a mass spectrum analysis is performed there is usually 500+ preliminary scoring peptides provided by correlating theoretical, reconstructed spectra against the experimental spectrum. This Inspector allows for analysis of all the output data from such analyses. The GUI provides dynamic re-ordering of the summary data, consensus calculations, removal of incorrect analyses, viewing of the raw spectral data via the IONgraph Inspector and other useful analysis methods.

The Visualize Mass Spectrum Inspector was constructed as the original proof of concept for Moulage System uses under the NIH Resource Center. It is currently being tested by one of the NIH collaborators. It has proven to be very efficient and useful. The CBP expects to be releasing this Inspector to all of the collaborators by 1st quarter 1997.