UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dottorato di ricerca in
Progettazione di materiali ad alte prestazioni
Nell'ambito della Scuola di Dottorato in
HIGH MECHANICS AND AUTOMOTIVE DESIGN & TECHNOLOGY
Ciclo XXIV

Development of a new generation of measurement instruments for the thermal analysis of materials.

Candidato:
Daniele Paganelli

Relatore:	Prof. Tiziano Manfredini
Relatore Esterno:	Dott. Mariano Paganelli
Coordinatore del Dottorato:	Prof. Giorgio Poli
Direttore della Scuola di dottorato:	Prof. Antonio Strozzi

Abstract

The present Thesis is the summary of three years of research in the field of optical measurement instruments for thermal analysis of materials.

The work was developed in collaboration with Expert System Solutions, which produces original optical instruments since more than 20 years ago, for example Heating Microscopes, Dilatometers and Fleximeters.

After an in-depth analysis of the actual instruments and development perspective, the design of a software architecture extensible towards new calculation procedures and additional peripherals was recognized as priority.

An innovative software was developed for the parallel acquisition and elaboration of data coming from multiple heterogeneous peripherals, such as cameras, stepper motors, thermoregulators, high precision balances. The new platform, called Misura4, increases the time resolution to more than 10 images per second, and the image resolution to 2048x1536 pixel.

Advanced image analysis methods produce more accurate and repeatable results. In the microscopy fields, introduce rigorous mathematical criteria for the identification of characteristic shapes and simultaneous analysis up to 24 samples.

The sample thermoregulation was designed in order to allow arbitrarily complex and long thermal cycles. A physical model was formulated for the heating and cooling of the kiln, from which a new thermoregulation algorithm was developed, capable of anticipating temperature behavior, while keeping high adaptivity towards external interferences (e.g. eso/endothermic reactions). The new algorithm improves the correspondence between the requested temperature curve and the performed one, a fundamental performance for dilatometry measurements.

The resulting software platform introduces new kind of measurements, which will be deepened through extensive experimental tests.

Acknowledgements

The biggest tribute goes to my father, dott. Mariano Paganelli, who started the extraordinary activity of laboratory instrument *invention* more than 20 years ago. He continuously gave precious insights regarding material science, in particular ceramics, and an enthusiastic vision about needed or interesting measurement possibilities.

I wish to thank Expert System Solutions and all my colleagues, whom directly or indirectly contributed to this achievement in many ways.

To Paola

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Optical, non-contact thermal analysis proved to give exceptional insights for the characterization and engineering of existing and new materials. The simulation of industrial thermal treatments and the continuous measurement of samples properties allows to reconstruct reactions and transitions undergone in the real production environment. Interferences from the measurement system itself are minimized by leaving the samples free to expand, contract and even melt in the furnace.

The entire range of instruments produced by Expert System Solutions were studied and future perspective considered. Current scientific research literature based on ESS instruments usage, and direct customer requests, highlighted the need for a move forward in the general architecture of the measuring system.

The present work was an highly interdisciplinary research effort. A new generation of thermal analysis instruments was designed, starting from the control software. Development methods and modeling theories were evaluated, studied and applied to obtain an high-quality software package, which could open new research possibilities.

The thesis organization partly reflects the chronological path of the research activity:

Chapter 1: Introduction. The state of the art in thermal analysis of materials is briefly presented. An in-depth focus is dedicated to the current instruments produced at Expert System Solutions: heating microscope, vertical and horizontal optical dilatometers and fleximeter. A list of applicative studies and publications is reviewed, showing disparate fields where optical thermal analysis is being successfully applied.

Chapter 2: Software After a detailed study of features and flaws, the current instrument control software was recognized as the limiting factor of measurement versatility and reliability. The entire software platform was rethought from scratch: from a single-process and almost single-thread application to a parallel processing environment. Modern development tools and concepts were applied, adhering to the Continuous Integration method. A Client/Server model was followed, where the mission-critical Server is designed to be installed on a dedicated, embedded device. The Client part is completely independent and cross-platform, ideally run on user's

commodity hardware.

Chapter 3: The Server Architecture The Server implementation is thoroughly explored. The concepts at the root of network transparency, configuration management, multi-processing acquisition and peripheral discovery and communication are explained with the help of detailed modeling diagrams.

Chapter 4: The Client Architecture The Client implementation is much more compact and has a simpler structure than the Server's one. Thanks to the high availability of advanced open-source libraries, most of the tasks could be imported and slightly adapted. Some additional, specific concepts were introduced for network translation of Server's objects into Client space.

Chapter 5: Image Analysis Methods Image analysis methods were developed, utilizing the whole length of the sample profile to calculate shape properties. The analysis can now adhere to main International Standards for the determination of fusibility of disparate materials. Current standards were not considered as a final destination, but only as the starting point for new research. More shape properties were introduced, and their relation with characteristic material shapes considered. A new numerical property, Cohesion, was developed as an all-in-one indicator of material's shape evolution during heating.

Chapter 6: Temperature Control Temperature is obviously a fundamental physical quantity for a thermal analysis instrument. External industrial-grade PLC devices for temperature control were abandoned, in favor of custom algorithm directly implemented in the Server software. Notwithstanding their ubiquitous use also in laboratories and in scientific instrumentation, automatic thermoregulators were considered *too limited* in their setpoint curve configuration and, at the same time, *too broad* in their power emission automation. The new temperature control is developed starting from a way less general physical model of the furnace, but allows unlimited freedom in thermal treatment design.

Chapter 7: Conclusions New methods should now be extensively tested by collecting experimental data and doing comparative studies. The novel software frameworks now allows for easy integration of advanced analysis and simulation techniques, which are already being studied for a near-term implementation.

1.1 Heating Microscopy

Heating Microscopes, also called *Hot Stage Microscopes*, were commercially introduced 60 years ago, and they immediately gained an important role in material science and engineering thanks to their wide field of application [Anonymous, 2002].

Nowadays, these instruments are used in both industry and research in many sectors: traditional and advanced ceramic industries, the glass industry, power

plants, the metallurgical sector, and, generally, in all fields where the melting behavior of materials has to be determined. Heating microscopy techniques can be successfully applied to the direct characterization of glazes, glasses, frits, ashes, mold powders for continuous casting, coals, slags, etc. — during a heat treatment.

A full understanding of the physical–chemical behaviors of these materials is often out of reach because of their complexity. To cite an example, a ceramic glaze may contain more than 20 chemical elements. This makes it close to impossible to obtain meaningful results *ab-initio*, that is, beginning from the fundamental properties of the single components forming the material, especially when new phases develop during the process. Experimental methods prove to be most effective for the study of the behavior of these materials during firing, allowing a good comprehension of behavior in an actual industrial firing cycle, thanks to direct observation of the sample.

When heating microscopes first appeared, manual analysis was extremely laborious and was often limited to recording the *softening temperature*, giving very subjective measurements. Furthermore, with the old manual heating microscopes it was not possible to reach high heating rates. The new generation models are completely automatic and can simulate industrial heat treatments up to 1600°C, with heating rates as high as 80°C/min or even “flash” heating.

The hot stage microscope has been used for many years for the research on the glass behavior. Beside the classical determination of softening point and sphere point, the hot stage microscope can supply many more useful information, if the images of the softening glass are properly evaluated and the heating rate can be modified in a wide range. The image processing applied to the study of the glass behavior was developed by Expert System back in the 1991, and since then it was continuously improved.

Hot Stage Microscopy is now widely used in many field of research linked to the melting behavior of materials. Thanks to the improvement in the computational speed it is now possible to perform a more accurate image analysis, gathering and averaging many images in a very short period of time. This opens the way to a better understanding of the behavior of the material even under very fast thermal treatment. In order to achieve this goal it is was necessary to design a completely new algorithm for the analysis of the shape of the specimen, as it is extensively explained further on in this work of thesis.

1.1.1 Measurement Principle and Evolution

The idea of heating microscopy is simple: look at a specimen inside a furnace to see how it changes in shape. Of course it is not as simple as it looks at first sight: at temperatures higher than 800°C the specimen start emitting light and is difficult to see clearly its shape. Expert System was the first to offer an heating microscope with automatic control of the contrast, with no need to adjust the optical system from room temperature up to 1750 °C.

Figure 1.2: First MISURA HSM heating microscope, dated 1991, and a model from 1996

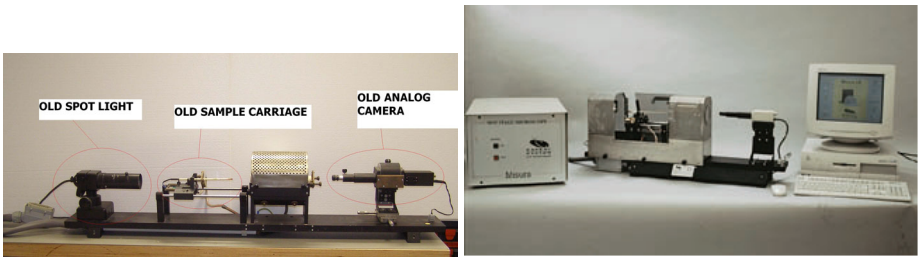


Figure 1.3: MISURA 2 HSM heating microscope dated 2000, and the current Misura 3 model (2012).

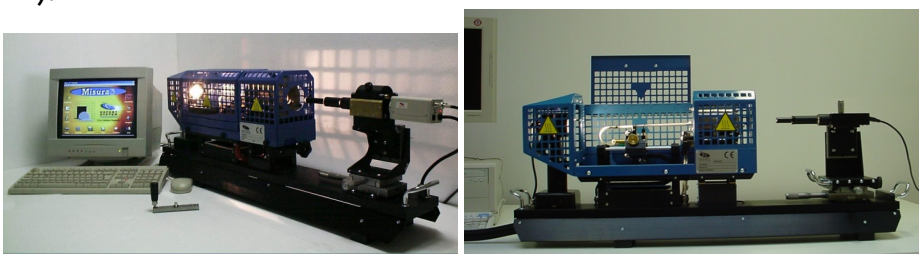
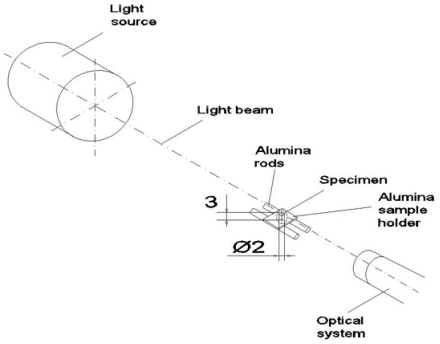


Figure 1.1: Heating microscope sample, light and optical system disposition



In glass and ceramic sector, heating microscope testing has never been ruled by a specific norm, even though since the first years of its introduction on the market (in the sixties by the German manufacturer Leitz), a standard analysis method was proposed by Sholze, but it never turned into a norm. The early studies of M. Paganelli in the eighties, proved that the practice of running laboratory tests with 3-mm high, 2-mm diameter samples at 50°C/min can provide valuable information about the behavior of the samples in industrial firing conditions.

In 1991 Expert System developed the first automatic heating microscope MISURA HSM with computerized image analysis and automatic temperature control up to 50°C per minute.

Since then, the instrument was continuously developed, passing from analogic to digital cameras, with improved temperature performances:

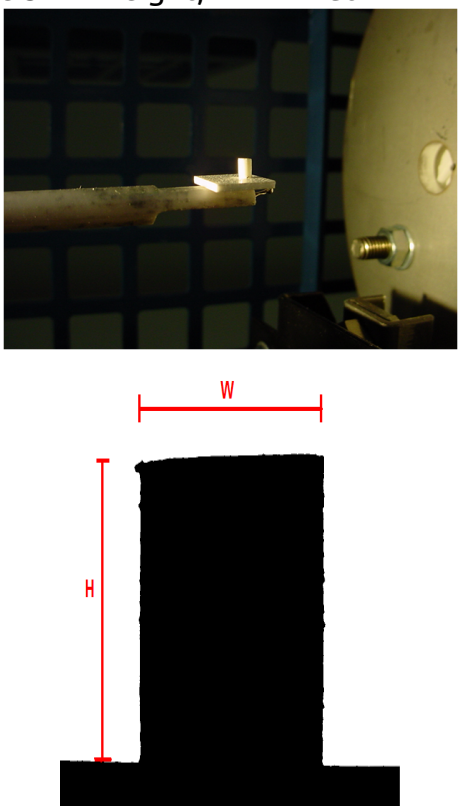
- up to 80 °C per minute
- with flash heating option
- up to 1750 °C with protective atmosphere

The proposed test method proved to be quite reliable because it gives reproducible results, even comparing industrial firing cycles with a typical laboratory cycle at a constant heating rate. The sample size of 3x2 mm was chosen considering the balance

between gravitational, viscous, and surface tension forces acting on the material during the melting process. The cylindrical sample was chosen because this is the only shape that enables the microscope to focus on a plane that does not change during the shape transformations induced by heating. The ratio between height and diameter is the best for the sphere point detection. The cylindrical samples (3 mm high and 2 mm in diameter) are obtained pressing the powders (after wetting with a drop of water) with a manual press into a cavity of suitable dimensions.

The characteristic points typical of glasses, glazes, and frits identifiable with the heating microscope are defined as following:

Figure 1.4: Best cylindrical shape proved to be 3mm height, 2mm width



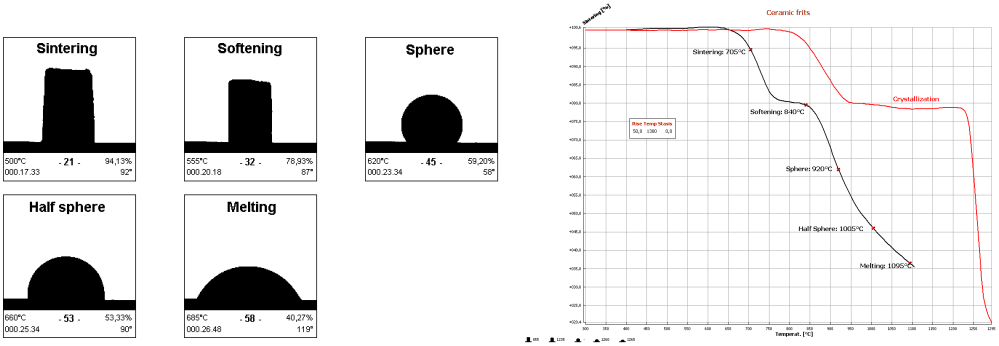
Onset of data acquisition: The image analyzer automatically takes as 100% the height of the sample, measured considering the sample holder (alumina support) as a base reference.

Sintering: As the temperature increases, the viscous flow activation threshold is overcome and the material undergoes the sintering phase: the sample decreases in size, but its shape does not substantially change. In the case of glasses, glazes, and frits, the driving force of the sintering process is the surface tension of the vitreous phases. The sintering phase ends when the sample reaches its maximum density. The sintering temperature identified by the instrument is the temperature at which the sample has reached a dimensional variation corresponding to the 5% with respect to the first image acquired, which is taken to be 100%.

Softening: The softening point is reached when liquid phases appear on the surface of the sample. From this point on, the shape of the sample undergoes substantial changes caused by the surface tension of the liquid phases. To find this point, the rounding of the corners of the sample and the smoothing of the upper part of the walls of the sample are taken into consideration.

Sphere: At the sphere temperature the sample is formed almost entirely of liquid phases, and the shape of the sample is controlled by the surface tension. While the surface tension tends to reduce the surface to a minimum forming a sphere,

Figure 1.5: Characteristic shapes and corresponding sinterization graph



the hydrostatic pressure related to the density of the liquid phase tends to flatten the shape. Some glasses do not reach the sphere point: for example, glasses with a high density and a low surface tension (lead silicates). In detecting the sphere temperature, every image analyzed is compared with a theoretical sphere.

Half Sphere: The half sphere temperature is reached when the height of the sample is half the width of the base. If the glass behaves normally, at the half sphere temperature the contact angle is approximately 90°. However, at this temperature the contact angle is often much higher, and in some cases the shape of the sample can resemble a bell. These anomalies can be attributed to the formation of non-homogeneous phases inside the sample, such as crystallization or glass-glass separations.

Melting: When the height of the sample shrinks to under a third of the base, it is assumed that the sample has completely liquefied and has reached the melting point.

Figure 1.5 shows on the left how those characteristic shapes look like.

The right plot compares two sinterization curves obtained for two different ceramic frits subjected to a heating rate of 50 °C/min.

1.1.1.1 Double optic heating microscope

ESS developed several version of multiple optics heating microscope to fulfill the requirement of special applications.

The left version in Figure 1.6 has two optics with different magnification, the right has two optics with the same magnification.

With the right model, called HSMx4, it was developed the possibility to perform simultaneous analysis over 4 different samples, as showed in Figure 1.7.

1.1.1.2 Flash heating

An important, recently introduced, testing implement is the FLASH heating cycle, provided thanks to a second thermocouple (independent from the samples holding system) and a motion controller on the kiln. The Flash tool allows to heat up the

Figure 1.6: Double optic heating microscope

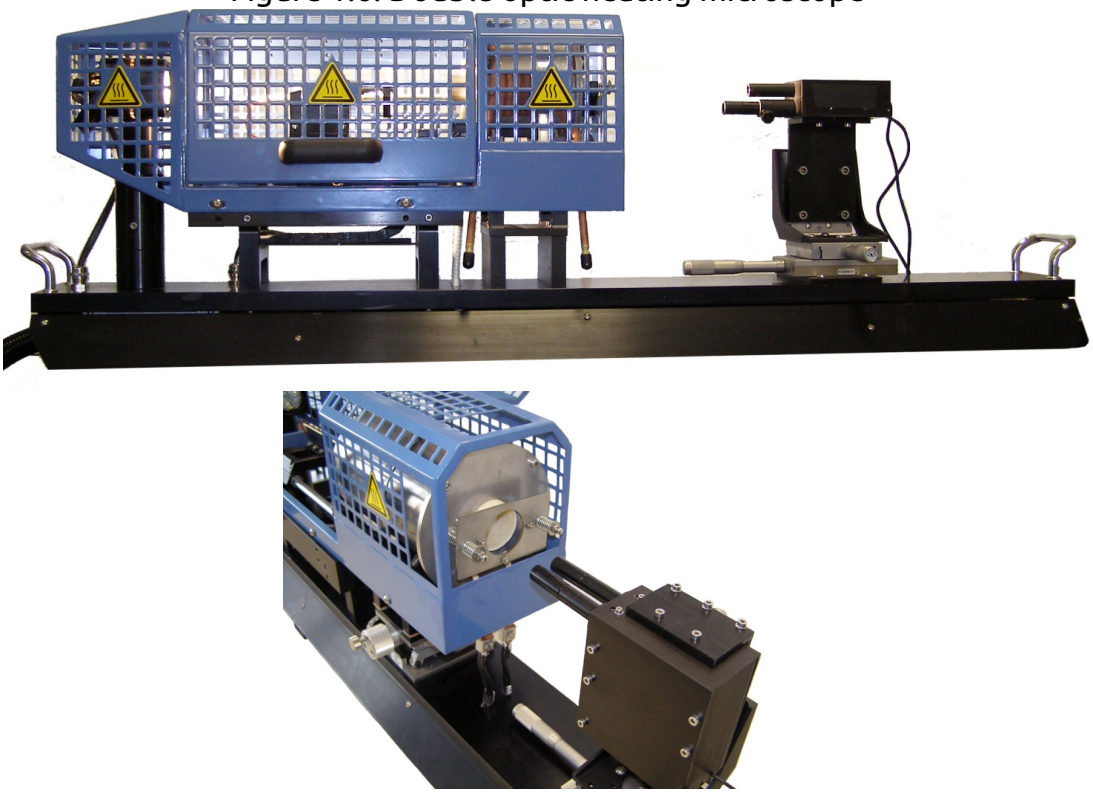
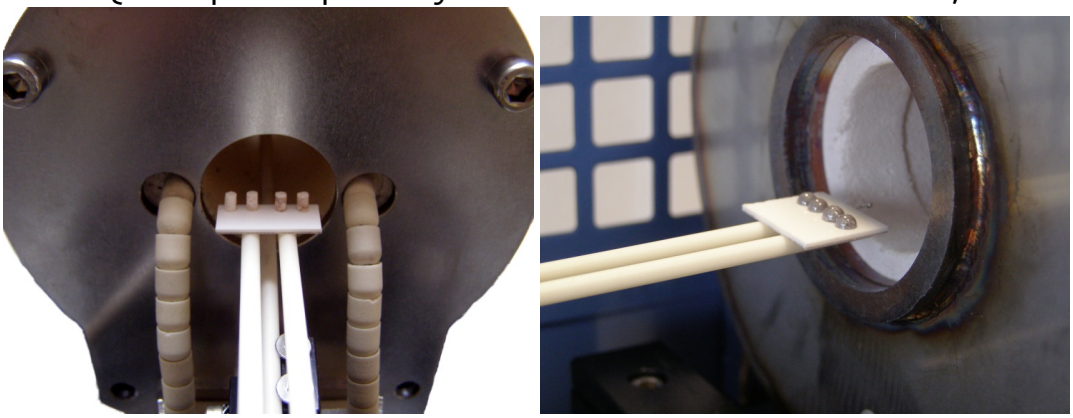
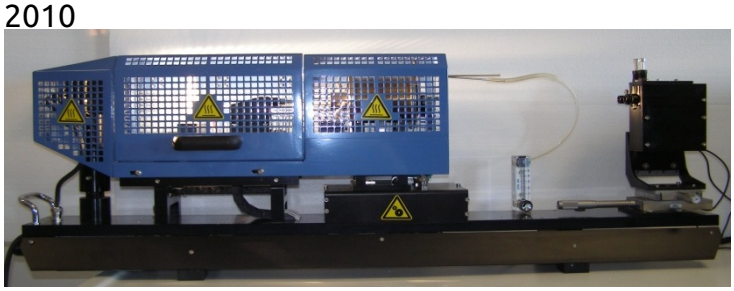


Figure 1.7: Quadruple sample analysis with the Misura 3 HSMx4 model, dated 2009



furnace at a given temperature and to slide it over the samples (already positioned and well framed by cameras) by means of an automatic motion control.

Figure 1.8: Misura 3 HSMx4 FLASH model, dated 2010



Samples are arranged on a 18 mm wide sample holder and are subjected to an instantaneous heating cycle during which the actual temperature on them is recorded together to digital pictures, taken every second and stored in a leased file for each sample.

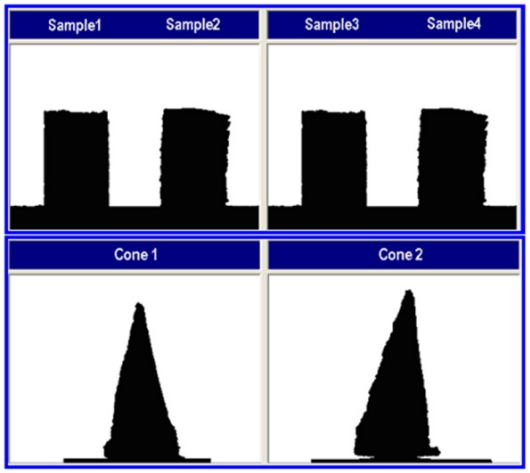
This system is able to reproduce extreme thermal stress conditions and some actual industrial thermal shocks can be simulated.

Typical applications are:

- Fuel Ash melting
- Powders for continuous casting
- Refuse derived fuels
- Industrial wastes
- Steel enamels.

1.1.1.3 Analysis of multiple samples of Fuel Ash according to the ASTM D1857 an ISO 540

Figure 1.9: Four Cylindric or Two Conical samples simultaneous analysis

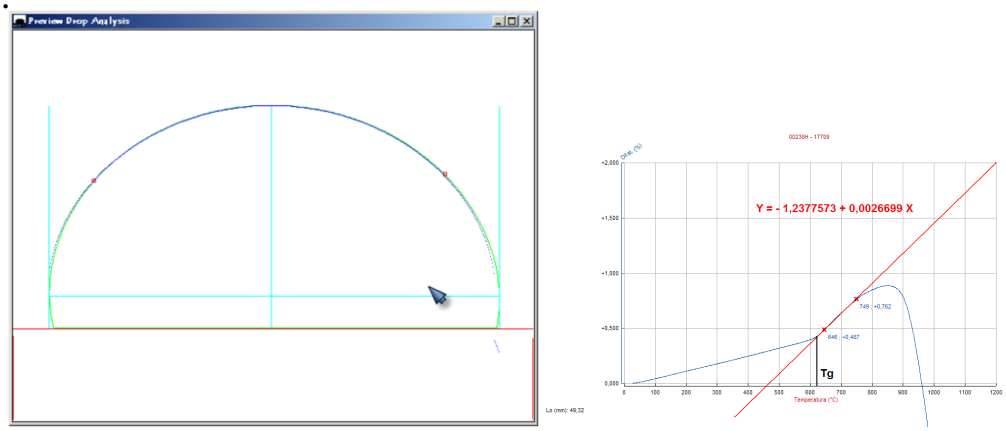


With the twofold purpose to fit the standard equipment with norms specifications for test specimens and to increase the number of samples simultaneously tested, ESS introduced in 2009 the new HSMN line. It can now avail of an enlarged kiln (200 mm in length and 35 mm as internal diameter), and different optics, specifically planned to entirely frame sample up to 19 mm in height (as specified from international norms).

Depending on sample dimensions (cylinder 3x2 mm, cube 3-7 mm in side, pyramid 11-19 mm in height) a specific optic is able to guarantee the maximum resolution for the measurement (different optical zoom for each optical path).

The American standard ASTM D1857 “Fusibility of coal and coke ash”, the International standard ISO 540 “Solid mineral fuels – determination of fusibility of ash”

Figure 1.11: Drop Runge-Kutta fitting and density extrapolation from dilatometry data.



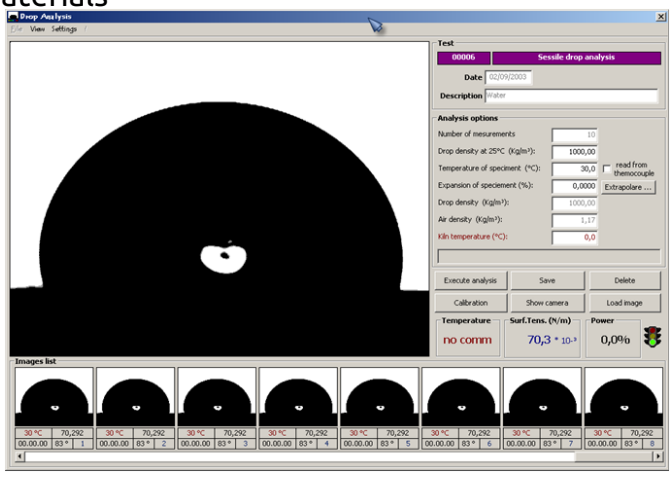
and the german standard DIN 51730 are the most common norms for ashes and use test pieces with sharp edges to facilitate the typical shapes observation (generally pyramidal shapes with height. up to 19 mm or cubic samples with up to 7 mm in side).

For standard small samples (like 3x2 cylinders or 11 mm height pyramids) is furthermore possible to use two side by side optics and increase the number of melting analysis per day. The software is able to manage two sample for each camera with separate automatic image analysis, so the technician can set up four tests at the same time.

1.1.1.4 Measurement of the surface tension on molten materials at high temperature

In 2004 Expert System Solutions implemented the measurement of surface tension based on the drop shape analysis using the Young-Lapalce equation on his heating microscope MISURA 3. The equations needed to solve the problem were developed step by step by several scientists, in the whole time lap of the last two centuries. The method that we applied is based on the Bashforth-Adams differential equation, which, unfortunately, has no known analytical solution. The solution is found applying the Runge-Kutta numerical integration and then minimising the mean square deviation between the calculated and the actual drop profile until the error falls below 10^{-4}.

Figure 1.10: Drop analysis on homogeneous molten materials



In order to get an estimate of the density at high temperature we compute the equation of the line passing through two points in the straight segment of the curve above the Tg, using the plot obtained with the double beam optical dilatometer. The almost straight segment above the Tg is the expansivity of the liquid glassy phase. Extrapolating this line up to the desired temperature we may calculate the increase in volume from room temperature to any given temperature. Since the mass of the sample is not changed due to the temperature increase, it is then possible to know the density at the given temperature. This make it possible to get reliable measurements of surface tension up to 1600 °C.

1.1.2 Hot Stage Microscopy Applications

The following paragraphs contains a literature review of most advanced and interesting applications of Hot Stage Microscopy. Cited studies were possible thanks to Expert System Solutions instruments.

1.1.2.1 Ceramic Glazes

The melting behavior of glazes and glass coatings affects surface smoothness and homogeneity, bubble evolution, crystallization, dissolution of colorants and refractory oxides, reactions between the body and under-glazes, and formation of the body–glaze interface. To improve industrial glazes, it is important to understand the melting behavior under industrial firing conditions. Hot-stage microscopy techniques can be applied for direct characterization of glazes, glasses and ceramic bodies during firing [Mushtaq Ahmed, 2003].

In the ceramic industry, a worldwide trend was to achieve a higher efficiency by changing to a firing technology with shorter firing schedules. This development has introduced new glaze formulations, often based partly or entirely on fritted formulations. Non-fritted formulations, raw glazes, are nevertheless, still attractive alternatives for many applications. The study of the behavior during the heat treatment using the Hot Stage Microscope is a very effective approach to study these new formulation [Fröberg, 2007].

1.1.2.2 Ceramic Frits

Typical formulation of fast firing glazes are made with fritted glasses which contains considerable amounts of zirconium and zinc oxides, which raise the production costs of ceramic glazes. Therefore there is a continuous effort to decrease these oxides in fast single-fired wall tile frit compositions and in industrial frit production. The study of the melting behavior can be accomplished using the Hot Stage Microscope which can reproduce the industrial firing cycle. This thesis deeply analyze a new algorithm for the temperature control, developing a new predictive model of the kiln [Pekkan and Karasu, 2009].

1.1.2.3 Porcelain Enamel

Another very active field of research is porcelain enameled steel composites, which involves an interdisciplinary approach between metallurgy and glass science [Silvano Pagliuca, 2011].

1.1.2.4 Glass

The effort to determine if alternative batching methods used in the commercial glass industry would result in improved melting behavior is a permanent task for the glass industry. Improved melting would enable shorter residence times in the glass tank, translating into direct reductions in energy intensity. Initial reactions between alkali and alkaline earth salts in the batch which create a low viscosity melt that segregates from quartz particles, resulting in a mixture that must then be re-homogenized in the tank, are being investigated with Hot Stage Microscopy [William M. Carty, 2003].

Bio-glass In bio-glass research the main concern is tailoring bioactive glasses to various clinical applications. The glasses should have a desired clinical performance and be restricted mainly by the requirements dictated by the site of implantation. The glass should thus show a certain bioactivity, described as attachment to tissue, reaction, and dissolution rates. Further, the glass composition should allow for high temperature working into desired products. This interdisciplinary field of research involve the use of hot stage microscopy to fully understand the behavior under melting and shaping of this kind of glass which has many compositional constraints given its unique field of application [Zhang et al., 2009].

Glass Filters Micro-porous ceramics have become increasingly popular in manufacturing filters for large-volume solid/liquid separation purposes, such as drinking, agricultural and waste water treatment, which require low-cost mass production of micro-porous ceramic filters with the desirable properties. The ceramic filter medium must have a high porosity, a narrow pore size range, and high bend strength, as well as high performance for the chemical nature of the filtered water. The constraints in the composition of these materials and the constraints in the industrial cost to make them suitable for the market, make it complex to find the proper formulation which must comply with the heat treatment needed for the manufacturing [Osman ana, 2008, Özgüra O., 2007].

1.1.2.5 Mould Powders

Mould fluxes are synthetic slag used to cover the liquid steel meniscus during continuous casting of steel and must fulfill several functions. They play an important role in the surface quality of the steel product and in the overall efficiency of the continuous casting process control. These powders have a critical influence on the heat transfer from the strand to the mould, the solidification process and the mould lubrication. The flux, which is continuously fed on the surface of liquid pool during casting, melts first and then flows into the gap between mold wall and solidified

steel shell. The study of the behavior of these materials pose some interesting tasks and it is accomplished using a hot stage microscope equipped with flash heating, which is able to reproduce the thermal stress of the flux powder poured onto the molten steel [Sighinolfi and Paganelli, 2011, Venturelli, 2011].

1.1.2.6 Fuel Ash

Advanced power systems such as integrated gasification combined cycle and fluidized-bed combustion (FBC) systems are at the forefront of power industry research because of the need for increased efficiency and reduction of greenhouse gases. Ash behavior in power systems can have a significant impact on the design and performance of these systems.

In the recent years the use of bio-fuels for power production has gained increasing importance as a substitute for coal or by being co-fired with coal. This development is particularly forced by the international concern about antropogeneous carbon dioxide emissions. However, the use of bio-fuels for steam raising or in gas turbine cycles is so far restricted by the fact that biofuels generally have a higher content of potentially deposit forming and corrosive elements than coal, especially on a heat value basis. The amount of melt present in an ash as a function of its temperature greatly influences the ash deposition propensity in thermal fuel conversion system. Therefore the study of the behavior of the ashes during the thermal cycle is becoming of increasing importance. The best proved way to perform this analysis is the hot stage microscopy with automatic image analysis [McCollor, 1997, Sighinolfi and Paganelli, 2009].

1.1.2.7 Fuel Cells

One of the forefront of energy research is now on the fuel cells and the Solid Oxide Fuel Cells (SOFC) are among the most promising. SOFCs work at relatively high temperature (900°C). Because of this high working temperature they pose severe problems of sealing and thermal stresses during warm-up and cool-down cycles. These problems can be approached using optical instruments like Hot Stage Microscopy and Optical fleximeter [Flugel et al., 2007].

1.1.2.8 Surface Tension

One of the most difficult parameters to measure on molten glass is the surface tension at liquidus temperature. Thanks to an improvement on Hot Stage Microscopy it is now possible to get accurate measurements of surface tension applying the Young Lapalce formula. This advance was presented at first by Expert System Solutions in 2003 [Paganelli, 2003a].

1.2 Double-Beam Optical Dilatometry

Measuring the thermo-mechanical behavior of soft material, or materials which become soft during the heat treatment, or ultra-thin materials has always been a challenge due to the low mechanical strength of very thin sheets. Even when reducing the force applied by the push rod of a standard electronic dilatometer to a minimum, it is nearly impossible to get a reliable measurement on sheets with a thickness below 100 microns because they bend too easily. Recently, a new instrument has been introduced specifically for measuring the thermal expansion of ultra-thin materials. The instrument can carry out thermo-mechanical measurements without making any contact with the sample, providing extremely accurate results. Additionally, the test procedure is designed to be easy and hassle-free, with no need to run calibration curves.

The double-beam optical dilatometer was first introduced by Expert System Solutions in 2003 and it is based on a new concept. Two microscopes with a very long focal length use blue light to illuminate both ends of a specimen placed inside the dilatometer's furnace. Since the specimen is very thin and unable to stand alone, the sample is placed horizontally on a special sample holder. The blue light used to illuminate the specimen has a wavelength of 478 nm which enables an optical resolution of 0.5 micron. The technology of image acquisition is rapidly evolving and it is now possible to reach sub-pixel resolution. Further on, in this work of thesis, the protocol for the sub-pixel resolution is investigated and a new solution is presented [Paganelli, 2004].

1.2.1 Measurement Principle and Evolution

The early attempt to push the optical resolution of a heating microscope up to the point to be able to measure the changes is size of the specimen due to the sintering process, were accomplished increasing the magnification of the optic of the microscope.

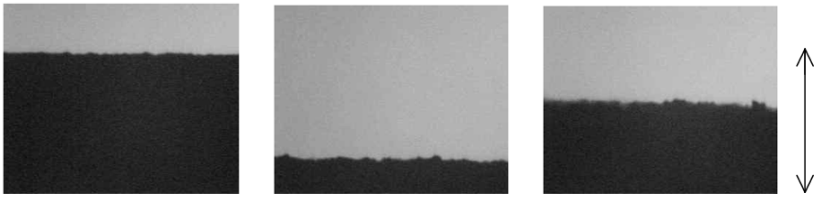
The first results were promising, but they were spoiled by the drift of the sample holder.

Using calibration techniques it was possible to correct the drift, but it was necessary to run a calibration curve for each time-temperature profile. Nevertheless the results were encouraging: for the first time it was possible to follow with high resolution a sintering process up to bloating (Figure 1.12).

In order to overcome this problem, it would have been necessary to measure at the same time the position of the sample holder and the position of the specimen. These studies lead in 2001 to the development of a double optic system. The two optical paths were driven onto the same CCD using an array of prisms. At that time, the size of the cameras was still far too big to think at two independent optical systems (Figure 1.13).

In Figure 1.14 the left the image of the two ends of the specimen, side by side on the same CCD, while the right plot shows the resulting sinterization curve.

Figure 1.12: Following profile border position during sintering.



Profile of the specimen during the heating. It is apparent that the surface becomes rough at maximum sintering. The arrow on the left rapresents 500 µm

Figure 1.13: Double optics vertical dilatometer

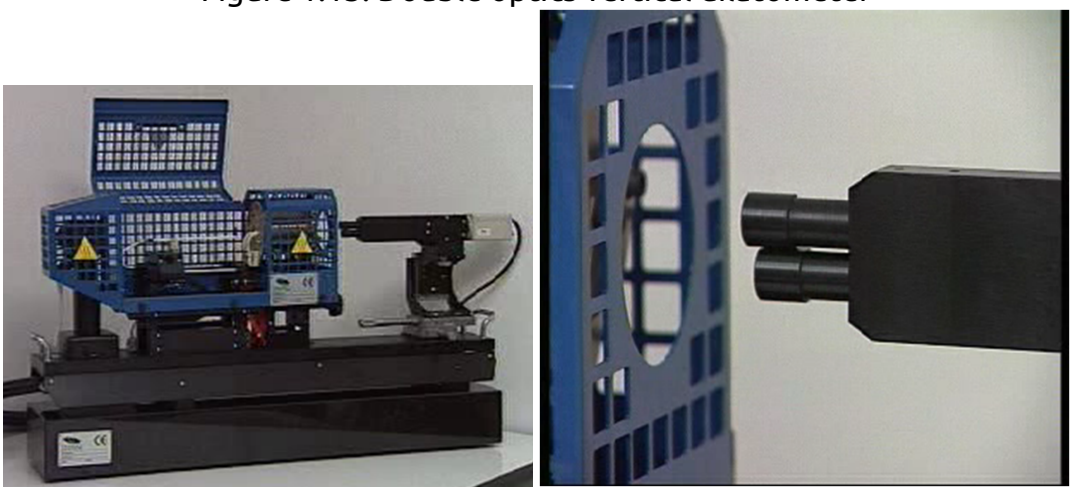


Figure 1.14: Following both sides of a sample over a single CCD.

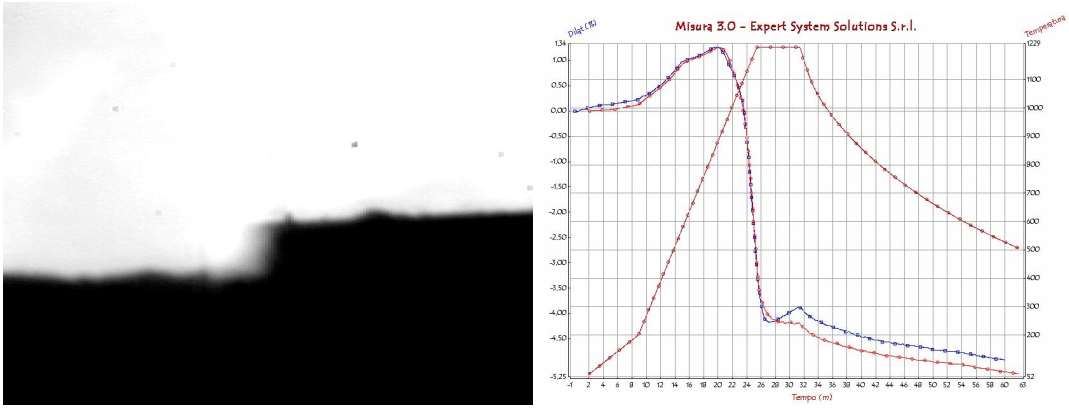


Figure 1.15: Following both sides of a sample with independent optics.

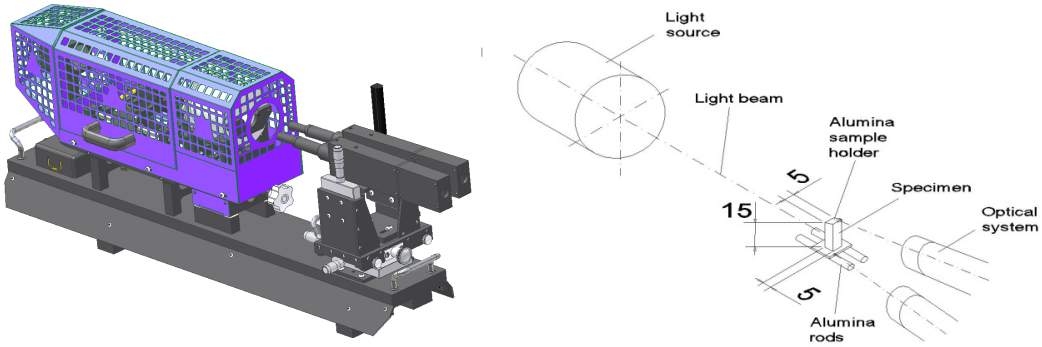
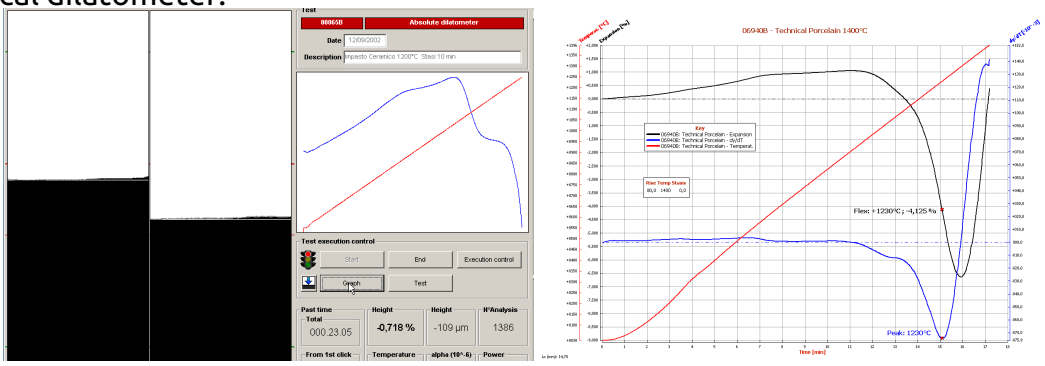


Figure 1.16: Automatic control software and sintering graph obtained from the vertical dilatometer.



This system had severe limitations: the specimen must be of exact size and the sintering can be followed only in a narrow range. But the results were impressive. It was possible to follow the change in size of the specimen with dilatometric resolution and with no contact. It was possible to see even small differences in the sintering behavior.

As soon as the new generation of miniature digital camera became available, Expert System developed the instrument with the two independent optical paths under the control of a step motor, in 2002 (Figure 1.15).

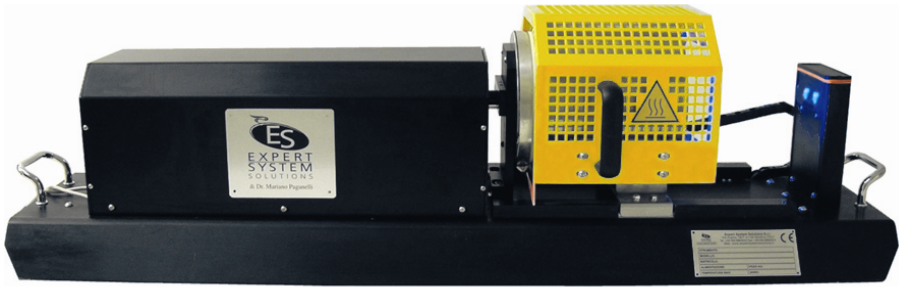
This instrument is also based on the double beam concept, but the two optics are now independent.

The first optical path is used to measure the top of the specimen (a step motor keeps it always aligned with the top of the sample).

The second optical path is focused on the sample holder (it is a reference beam to correct the mechanical drift of the sample holder).

Any drift is instantly corrected from the actual measurement. Thus measurements are absolute, since they are not affected by the sample holder or by any push-rod, as in traditional dilatometers. There is no need for the calibration curve and it is possible to change the heating rate or to design a complex heating curve. Figure 1.16 shows a screenshot and a plot of a sintering curve up to bloating.

Figure 1.17: Misura ODLT horizontal double-beam optical dilatometer, 2003.

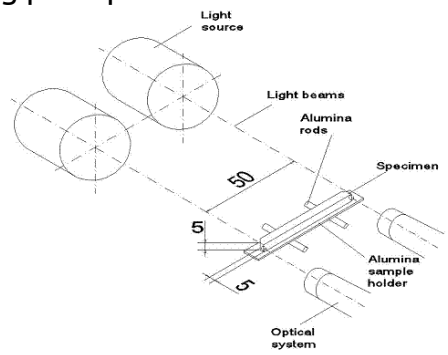


1.2.1.1 Horizontal double beam optical dilatometer

Measuring the size of the specimen using a beam of light has an intrinsic limitation: the wavelength of the light.

In order to improve the resolution of the measurement, the only possibility is to make the sample longer. This work lead to the development of the horizontal dilatometer, which is designed for 50 mm long specimen and can reach a resolution of 1 part over 100.000 (Figure 1.18).

Figure 1.18: Horizontal dilatometer measuring principle.



The use of two beams of light implies the use of two microscopes and two digital cameras. Focusing the image of the tip of the sample on the CCD of the camera with the maximum of the magnification achievable using visible light, means to be able to see only few hundred microns of the sample.

A sample 50 mm or 50.000 microns long which reaches an expansion of 1 % will expand 500 micron. Using blue light with wave length of 478 nano meters and magnifying up to 0,5 micron per pixel, the image will shift 1000 pixel. Since the sample is free on the sample holder, it may move in one direction only, going out of the field of view of a video camera with 1 megapixel (1000x1000). The use of line cameras is unfeasible because the measurement will be strongly affected by edge imperfections. The use of larger area CCD implies to be able to obtain a perfectly flat filed over a large area CCD, and working with this magnification is close to impossible.

The solution to this problem is to move the optical path in order to follow the expansion or the contraction of the sample. It is necessary to use two linear translation stages equipped with a step motor controlled by the computer and able to make the shift in a very short time with very high accuracy. This new design, patented by Expert System Solutions, allows to build optical dilatometers with unprecedented features: using a sample 50 mm long placed horizontally in the furnace, it can measure expansion with a resolution of one part over 100.000, or using a sample

15 mm long placed vertically inside the furnace it can measure sintering or bloating process with no limitation in dimensional changes.

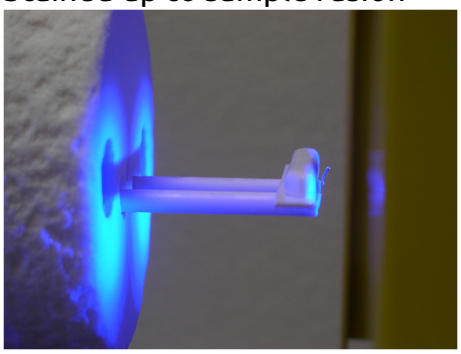
One relevant feature of this kind of dilatometers is the fact that the calibration curve it is not necessary. Once the magnification of the two optical paths is carefully established, there is no need for further calibration, unless the instrument is mechanically taken apart. In fact, the magnification of the optical system do not change with time or with the number of test performed.

A potential problem with this measuring method is that the sample is free to expand in both directions. In some cases, a negligible amount of friction between one side of the sample and the sample holder might cause the actual expansion to occur only on the other side of the sample. As a result, the image of the edge of the sample that is expanding can go out of the field of view of one camera due to the extremely high degree of magnification. However, this problem can easily be corrected by moving the optical path of both cameras with a linear motor to bring the edge of the specimen back into the camera's field of view. The displacement is registered electronically, and the final data plot is completely seamless.

Measuring both ends of the specimen at the same time without making contact provides an absolute measurement.

Measuring Up To Melting Thermal expansion measurements on glass have traditionally been performed using push rod dilatometers where the contact between the sample and the measuring system does not allow to get high quality expansion data in a wide temperature range. In order to obtain highly reliable results in measurements above the glass transition range (with a minimum effect associated with the viscous deformation of samples), it is necessary to use a dilatometer with a weak measuring force and samples with a large cross-sectional area, so that the measuring pressure is kept at low values. The Horizontal Dilatometer Misura® ODLT solves completely these problems thanks to the optical non contact measuring system.

Figure 1.19: Thermal expansion data can be obtained up to sample fusion



The optical dilatometry is attaining great reliability on a large number of technological fields and especially for materials undergoing viscous flowing during heating; glasses represent therefore an ideal application to outline new laboratory testing possibilities (Figure 1.19).

Thin and ultra thin specimens Low temperature co-fired ceramics (LTCCs) are used for the production of high-integrated multilayer circuits (Figure 1.20).

A flexible raw material tape obtained with the doctor blade tape casting process is used as a base. During the co-firing process, the binder burnout takes place.

Figure 1.20: Doctor blade tape casted thin foil and its sintering curve.

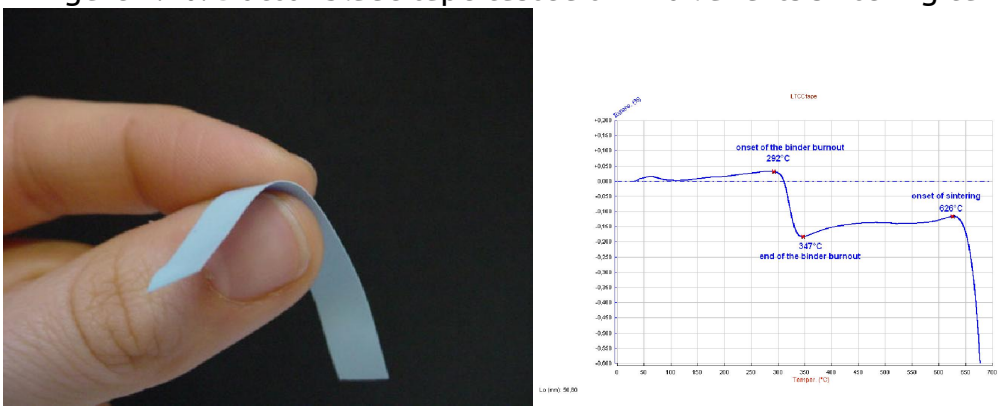
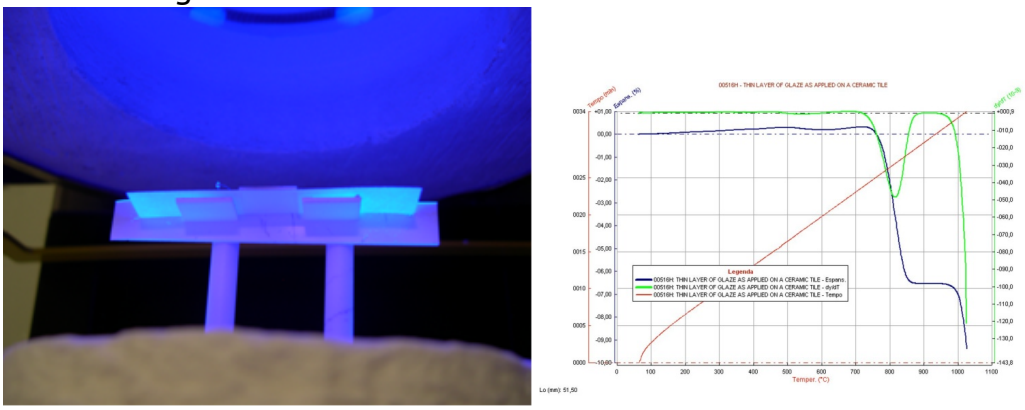


Figure 1.21: Ultra thin specimens of the unfired glaze layer on a ceramic tile. On the right the sintering curve.



As the temperature increases, the glass phase present forms a viscous melt at low temperatures (below 900°C), that activates liquid phase sintering. The molten glass acts as a binder for all the crystalline phases present. The densification and the shrinkage of the substrates occur. The presence of a liquid phase during sintering makes the material substantially viscous.

1.2.1.2 Differential Thermal Analysis setup inside the ODLT-DTA

In year 2007 Expert System Solutions introduced the optical dilatometer combined with the Differential Thermal Analysis. Taking advantage of the double rod sample holder, we installed a thermocouple in both rods. Placing a platinum crucible (made by bending a thin platinum foil) onto the thermocouple tips, it possible to record the temperature differences between the specimen and the reference material, performing a Differential Thermal Analysis.

As shown in the Figure 1.22, it is possible to use the same instrument for recording the sintering curve and the DTA at the same heating rate (but not on the same specimen at the same time).

Figure 1.22: DTA on a glass ceramic material and a a graph displaying both sintering curve and DTA result.

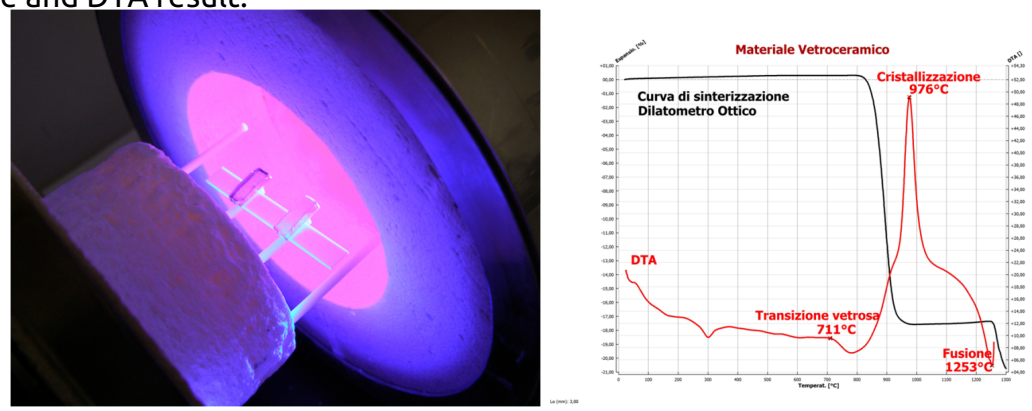


Figure 1.23: High performance Horizontal Dilatometers

1.2.1.3 High performance horizontal dilatometer

The need to improve the performance of the horizontal dilatometer lead to the development of the high performance double beam horizontal dilatometer, with near zero hysteresis 2009 (Figure 1.23, on the left). This result was achieved thanks to the following implements:

- Granite base for improved rigidity
- Heat shield between the kiln and the optical bench
- Optical bench under thermostatic control .

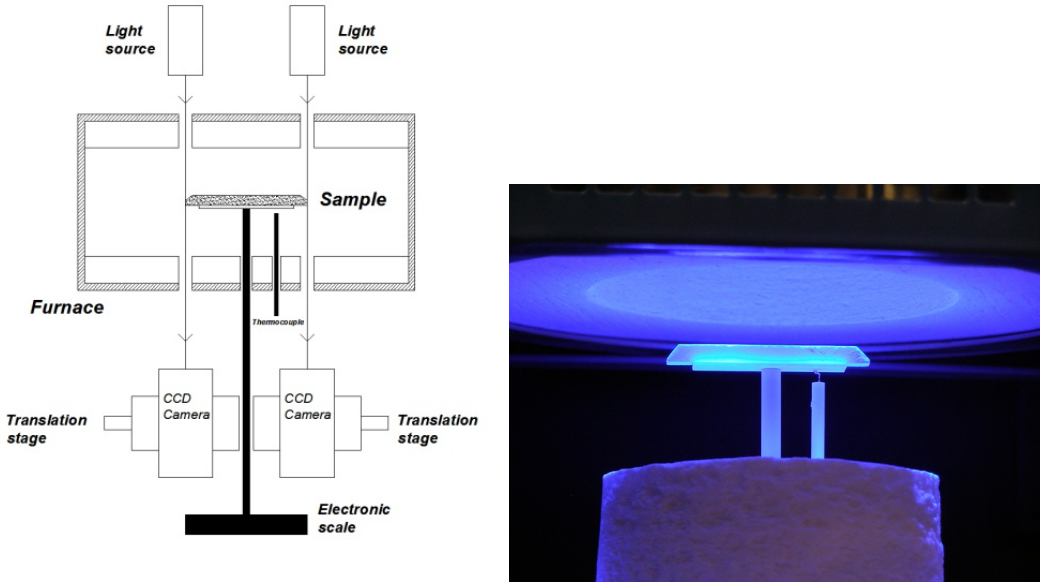
Thank to this development it is possible to reduce the hysteresis in cycling tests (heating and cooling) down to a few microns.

The use of molybdenum disilicide heating elements permitted to achieve another exceptional result in 2010. The maximum temperature of the horizontal dilatometer was increased up to 1750°C (Figure 1.23, on the right), and the instrument combined both dilatometer and heating microscope on the same chassis.

1.2.1.4 Simultaneous thermal expansion and thermo gravimetric analyzer

Thanks to the fact that the measure is carried out with no contact it becomes possible to carry out other measurements during the test, like the change in mass.

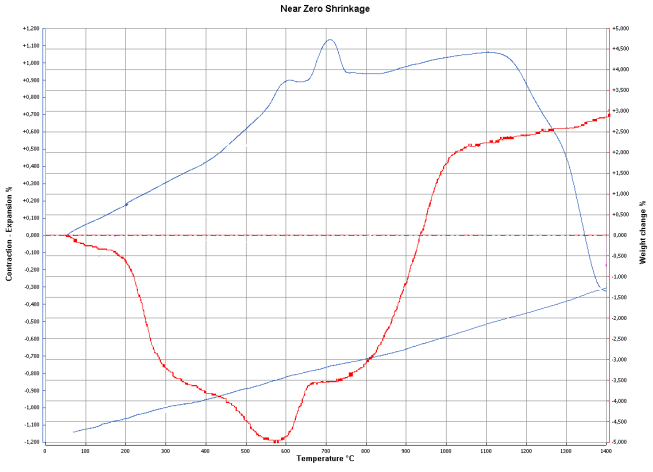
Figure 1.24: High performance Horizontal Dilatometers



Joining a double beam optical dilatometer and an electronic scale we can measure simultaneously the change in size with a resolution of one part over 100.000 and the change in weight of the specimen, with a resolution of 1 part over 2500.

The scheme of the instrument is displayed in Figure 1.24.

Figure 1.25: Change in size and change in weight on a iron rich high silica slag.



The specimen exceeds the length of the sample holder, in such a way that the two ends remain in the field of view of the two cameras. The sample holder, through a vertical alumina rod, is connected with the electronic scale at the bottom of the instrument.

The furnace has four viewing windows: two on the top, which enable the light from the light sources to illuminate the ends of the specimen, and two on the bottom, through which the ends of the specimen are framed by the two digital cameras.

The two digital cameras are mounted on translation stages, controlled by microstepping motors. This innovative design allows the simultaneous measurement of changes in mass and changes in size (Figure 1.25).

The specimen can be 50 mm long and 5x5 mm in cross section. Considering a green density of 2 g/cm3 the total weight will be around 2,5 g. Using a scale with 1 mg sensitivity we get a resolution of one part over 2500. The resolution can be

raised up to one part over 25000 if the scale is 0,1 mg sensitivity.

Furthermore, this new development goes beyond the limitation of the traditional temperature controller for the design of the heat treatment. The heat treatment can be designed with complete freedom, even from a mathematical formula (i.e. sinusoidal or exponential) and it can be controlled by the change in size of the specimen during the test. This innovative feature makes it possible to perform the Controlled Rate Sintering, as far as we accept the compromise to measure the linear shrinkage in place of the true sintering as percentage of the theoretical density.

1.2.2 Optical Dilatometer Applications

This subsection presents many studies conducted with the help of the optical dilatometer. Verbatim quotation is extensively used to introduce the topic of such disparate fields of application.

Soft and flexible materials are becoming quite common also in the refractory industry. The fact they are flexible greatly reduce the thermal stresses during heating and cooling, but makes also quite difficult to measure their thermal behavior. Thanks to the double beam optical dilatometer it is now possible to make accurate analysis of thermal behavior even without touching the specimen [Chiara Venturelli, 2006, Paganelli, 2003b].

Beside these forefront application of the thermal expansion measurements, the most of the practical application of dilatometry is in the measurement of the coefficient of expansion (COE) on solid materials. Thanks to the new double optical dilatometer, the ceramic scientist has the possibility to follow the behavior of the sample during the heat treatment without interfering with the process. The applications brake the limits of traditional dilatometry in many fields of research:

- Incoherent materials, like the expansion and contraction of the raw glaze applied on the surface of a ceramic ware, or the behavior of an incoherent granular frit
- Softened materials, like the behavior of a glass above the transition temperature, where the surface tension starts to pull the edges and to make the sample shorter
- Sintering kinetics: studying the relationship between time and temperature during a sintering process
- Extremely thin samples, like tape cast substrates can be analysed with regard to thermal expansion and sintering behavior [Davide Sighinolfi, 2010c].

1.2.2.1 Foundry Slag

Glasses and glass-ceramics are suitable media to immobilize regulated heavy metals and recycle inorganic hazardous wastes. During the production of the sintered glass-ceramic, the formation of the crystalline phases may influence the sintering process

and the use of an optical dilatometer enables the possibility to measure the true change in size or volume of materials during the process. An innovative simultaneous dilatometry / TGA investigation is under development by Expert System Solutions, in order to get more reliable data on the sintering kinetic of these glass-ceramic systems [Davide Sighinolfi, 2010a]. The full development of this unique research tool for the material engineering require the development of a new protocol which is discussed in this thesis work.

1.2.2.2 Sintering of Traditional Ceramics

Traditionally sintered ceramic bodies, such as stone-ware or porcelain-ware, undergo a viscous flow sintering process. The driving force is mainly given by the surface tension of the liquid glassy phase, and the speed of the process is controlled by the viscosity of the glassy phase. During the past few decades, the concept of fast firing has gained more and more acceptance in the industry. This has led to the need for development of fast-sintering ceramic bodies that can yield a fully vitrified ware with a total firing cycle of <30'. The most important factor to increase the sintering speed in a viscous flow sintering process is to decrease the viscosity and to increase the surface tension of the glassy phase. A major concern in the design of fast-sintering bodies is the determination of the best top firing temperature, which is the temperature at which the given body composition is able to achieve full vitrification, with no bloating, in the minimum time. This is not a trivial problem to approach, because it implies the ability to follow the sintering process without introducing perturbation. The double-beam optical dilatometer is the ideal instrument for this task. It can follow, with dilatometric resolution, the sintering of the material with no contact, directly during a fast-firing cycle [Paganelli, 2002].

The possibility to realize bodies with different dimension for the quartz and feldspatic portion is not even very far from the industrial reality in which the technological evolution brings innovative grinding systems which allow the separate comminution between the clay fraction and the hardest, toughest and most insoluble components. One of the industrial problems yet to be solved is the evaluation of quartz thermal inertia degree in accordance with its particles grain size. The dimensional limit concerning the grains of quartz affect the sintering during firing. This very important industrial parameter can be deeply investigated using the double beam optical dilatometer, which is able to follow the sintering process with no mechanical interference [Davide Sighinolfi, 2009].

1.2.2.3 Sintering of Glass-Ceramic

Glass-ceramics were first investigated in the 1940s by Stookey at Corning Glass. Since the 1940s, many systems have been extensively studied throughout the world. Because of their specific set of properties, such as good chemical and mechanical resistance, glass-ceramics have found industrial applications, military applications, and household applications such as in cooktops, cookwares and bakewares. In addition, they have been used in protection layers, sewing-thread supports in the

textile industry, and ceramic tiles.

Now, thanks to the research possibilities offered by the non contact optical dilatometer developed by Expert System Solutions, it is possible to follow the sintering process of a glass-ceramic powder or the volume change in the phase transformation between glassy phase and crystalline phase [Klegues et al., 2009]. This broadens the research field and opens new scenarios for the glass ceramic industry.

1.2.2.4 Secondary Raw Materials

Currently porcelain tiles are regarded as the best quality product the ceramic tile industry has ever developed. The production process is almost similar to that of traditional ceramic tiles. Porcelain tiles are obtained as a dense material comprised of a larger amount of glassy phase with quartz and mullite crystals after being sintered at 1210 °C or 1215 °C .

However, there are some technical and economic difficulties associated with the production of porcelain tiles owing to the necessities for high purity raw materials and a larger amount of glassy phase in the bodies in order to obtain very low water absorption values in their microstructural design, compared to traditional floor and wall tiles.

These necessities have made the investigation of alternative raw materials inevitable and there have been numerous researches on alternative raw materials to be used in porcelain tile production. Research has shown that alternative sintering materials should not only have the appropriate chemical and mineralogical compositions but also interact appropriately with the other constituents in the body in terms of thermal character and microstructural phase transformation property at the sintering stage which are determined by characterization techniques such as an optical dilatometer.

Boron oxide-bearing minerals can technologically be considered as important alternative raw materials for porcelain bodies since they have a glass-forming character and have been used in the ceramic industry to reduce the melting point of the glass phase such as in the production of frits, glass [Klegues et al., 2009].

Ceramic Sludges The ceramic residues coming from tiles industry are generated in the process and purification treatments and can be classified in two different ways as a function of both their production process and physical state. One group is composed of unfired rejects coming from steps prior firing (pressing, drying, glazing, etc.) and the other one consists in fired rejects, derived from tile selection, polishing, etc. At present, the current industrial reintroduction of by-products into the same process is the best viable solution to combine environmental (reduction of dumping effluents and saving water consumption) and economic benefits [Fernanda Andreola and Lancellotti, 2010].

Solid Wastes The generation of municipal solid waste (MSW) in Italy in 2007 was 32.55 million tons and the amount of incinerated waste was around 4.5 million tons.

Solid outputs of incineration are represented by fly and bottom ashes that in the case of municipal solid waste incineration (MSWI) ranges from 3–5 to 20–35% by weight of the original quantity of waste for fly and bottom ash, respectively . The bottom ash seldom contains significant levels of heavy metals. While fly ash is always classified as hazardous waste, bottom ash is generally considered safe for regular landfill, after a certain level of testing defined by the local legislation. Many efforts have been made to improve the environmental quality of residues from waste incineration and to recycle or utilize at least part of specific residue flows. The recovery of waste matrices otherwise destined for disposal allows to save resources, reducing the need for natural raw materials. Considering that recycling has priority over disposal/deposition in landfill sites and the use of secondary raw materials reduces costs and conserves resources, it is a need in the processing of waste materials into usable, quality-assured secondary raw materials (SRM).

In the last years in Italy some specialized companies in post-treatment technology of bottom ashes were developed. The objective of minimizing the use of natural raw material increasing the use of secondary raw materials, such as bottom ashes, is being approached studying the sintering process with the non contact optical dilatometer [Nezahat Ediza, 2009].

Cathodic tubes Nowadays, recycling processes is becoming more and more important mostly due to the impressive increase in the production of wastes and to the growing attention to the environmental safeguard.

In modern society, the improvement of life standards and the technological development have brought about a significant growth in the consumption of computer (PC) and TV-sets. The consequent increasing of wastes, coming from electronic and electrical devices, requires to make their recycling technically and economically feasible .

Currently, the level of reutilization of both PC and TV wastes is about 5 wt.% of the total production, but the latest trend is moving towards an increasing of these levels; for this purpose, recent investigations within European countries estimated the necessity to allocate about $10^6 \frac{m^3}{year}$ of TV glass wastes per million people. The 85 wt.% of PC and TV waste is made up of cathodic ray tubes and screen glasses, and, notwithstanding their particular chemical composition due to the presence of some hazardous elements, it has been demonstrated that their recycling on an industrial scale is possible.

The introduction of TV and PC cathodic tubes is being exploited and the laboratory research made possible by the optical dilatometer [Schabbach et al., 2011].

Industrial wastes Nowadays there is also a lot of research aimed at exploiting the peculiar features industrial wastes which would otherwise be disposed. Many studies about granite and ceramic tile sludges are in progress, with the aim to produce lightweight expanded aggregates with technological properties suitable to get lightweight structural concretes with improved thermal performances, matching the ever stricter new requirements in the building and construction field. The

swelling behavior needed to obtain lightweight aggregates is measured using the non contact optical dilatometer [R. de Gennaroa, 2009].

1.2.2.5 Ashes

In Europe the amount of coal combustion products (CCPs) produced in 2003 totalled 65 million tonnes, 68% of which was pulverised fuel ash (PFA). The disposal of this by-product is therefore a significant economic and environmental burden and there is an increasing need to develop novel, economically viable and environmentally sound alternative reuse applications.

Extensive research has investigated the use of PFA in glass- ceramics and some work has also been completed on ceramic tile manufacture using PFA.

A 2006 study on the effect of the addition of borates on the properties of sintered PFA [Uwe et al., 2007] utilized the double-beam optical dilatometer in order to characterize the sinterization process.

1.2.2.6 High-tech ceramics

The non contact optical dilatometer finds many uses also in high tech applications. One example are the low temperature co-fired ceramics (LTCCs), which are used for the production of high-integrated multilayer circuits.

A flexible raw material tape obtained with the doctor blade tape casting process is used as a base. On this non-sintered green film, which consists of a mixture of glass-ceramic composites, organic binders and solvents, are applied some conductive, dielectric and resistive pastes. The sheets obtained are laminated together in a large number and, finally, fired in one single step.

During the co-firing process, the binder burnout takes place. As the temperature increases, the glass phase present forms a viscous melt at low temperatures (below 900°C), that activates liquid phase sintering.

The molten glass acts as a binder for all the crystalline phases present. The densification and the shrinkage of the substrates occur.

The presence of a liquid phase during sintering makes the material substantially viscous. The only research tool able to follow the process of sintering in presence of liquid glassy phase is the optical dilatometer [Chiara Venturelli, 2005].

Another file of advanced application is the development of microporous substrate [C. Özgüra, 2011].

1.2.2.7 Porcelain Tile Decorations using Metallic Powders

Recently, in the never ending effort to develop an intriguing new look for porcelain tiles, it was developed the introduction of metallic powders in traditional porcelain stoneware bodies. This artifact makes the well known ceramic system a new complex composite material which thermal behavior during fast firing processes and final mechanical properties, should be taken under control. Innovative porcelain stoneware tiles with a surface layer containing stainless steel particles, are currently produced by the Double Charge Technology and considering this layer as a composite

Figure 1.26: Measurement principle.

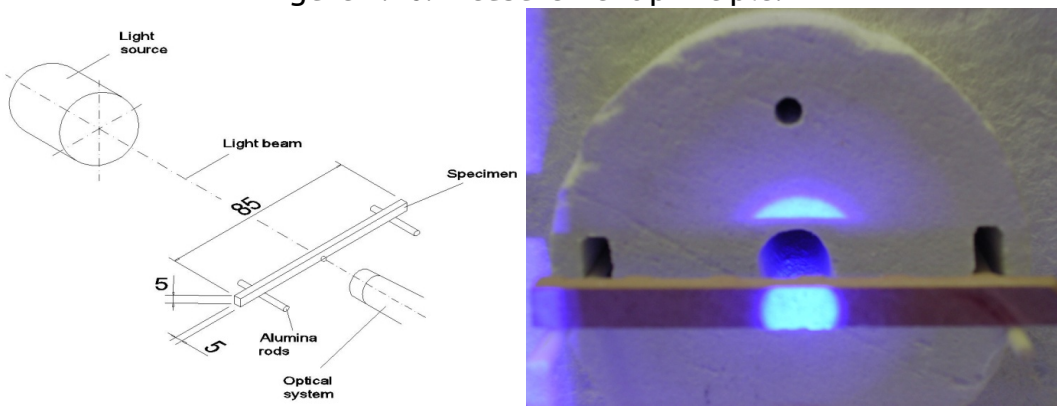
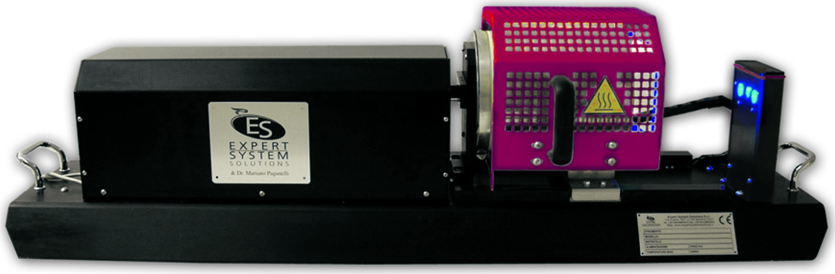


Figure 1.27: Misura Flex



material, it appear evident the importance of mechanical investigations such as Young modulus, fracture toughness and flexural strength. In particular, the study of the sintering behavior has been pointed out in order to clarify the role of the metal-ceramic interface and to investigate, with a contactless technology, effects of the oxidizing atmosphere of ceramic roller kilns on metal particles. The use of the non contact Vertical Optical Dilatometer enables a careful simulation of industrial firing cycles and represents therefore the ideal equipment for advanced experimental high temperature analysis [Davide Sighinolfi, 2010b].

1.3 Optical Fleximetry

The first idea of an instrument capable of a non contact measuring of the bending of a specimen under a heat treatment was developed in 2003. Using the same technology of imaging and positioning developed for the optical dilatometer, Expert System Solutions designed MISURA FLEX.

The basic idea is simple: suspend a specimen shaped as thin bar between to sample holders rods (Figure 1.26). Illuminate the bottom of the bar with a beam of blue light (478 nm) and frame a small portion of the bottom surface using a digital camera equipped with a high magnification optic with a long working distance. If the specimen is subject to a change in shape, bending downward or upward, the shift will be recorded. The position of the optical system is controlled by a step-motor, which moves the camera when the image of the sample is about to exit from the area of the CCD. This instrument proved to be very useful in several applications.

Figure 1.28: Misura Flex

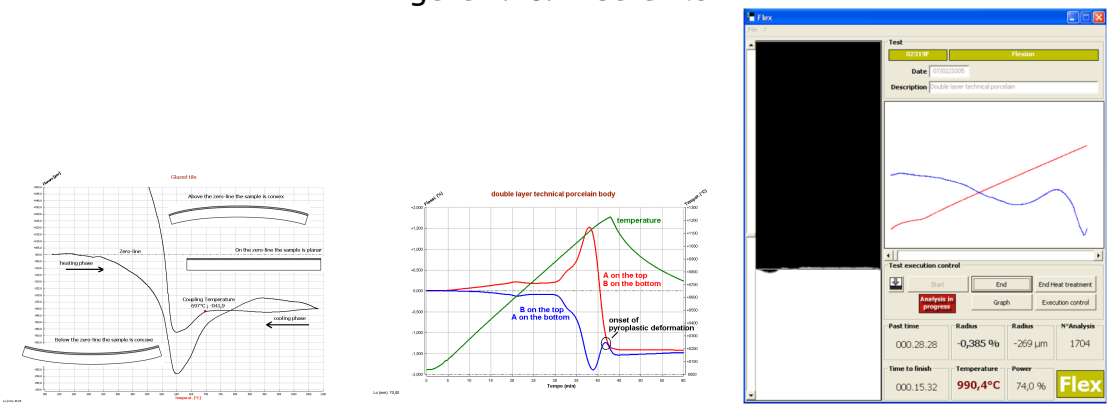


Figure 1.27 illustrate an instrument with three optical paths: the two external are used for the optical dilatometer, the one in the center is used for the optical fleximeter.

The interpretation of the curves is not obvious, even if at first sight, they are quite easy to understand: if the curve goes down, it means that the sample is bending downward, if the curve goes up, it means that the sample is bending upward.

It is against the odds that a bar of ceramic body during a heat treatment may bend upward, but this is what it may really happen if the specimen is made of two layers of different materials.

Another very interesting application is the study of the residual stresses in a double layer material, like already fired glazed ceramic ware. This test was proposed long time ago by Steger, but it was almost abandoned because of the intrinsic difficulty of the measurement. Now it can be performed easily with this automatic instrument. Residual stresses in double layer ceramic wares are responsible of many technological problems, like bending, scaling, crazing or easy chipping and cracking. Therefore this test is now gaining new acceptance.

One of the biggest technological problems in the today ceramic industry is the geometrical perfection of tiles which are every day larger and thinner. One of the best way to approach the problem is to study the pyro-plastic deformation. Thanks to the optical fleximeter it is possible to measure the speed of deformation which is directly linked to the viscosity of the glassy phases which develop in the ceramic body during the firing.

The graph on the right of Figure 1.29 shows the comparison of two body composition. The dotted lines are the speed of deformation. It is surprising how different the two behavior are, considering that both the materials are taken from industrial samples. The study of these behaviors leads to the optimization of the technology to manufacture perfect big size tiles.

A new interesting filed of application is the study of the deformation upon wetting (during the glazing) of the green ceramic ware.

The study is carried out at room temperature wetting with a controlled amount

Figure 1.29: Deformation speed amongst materials.

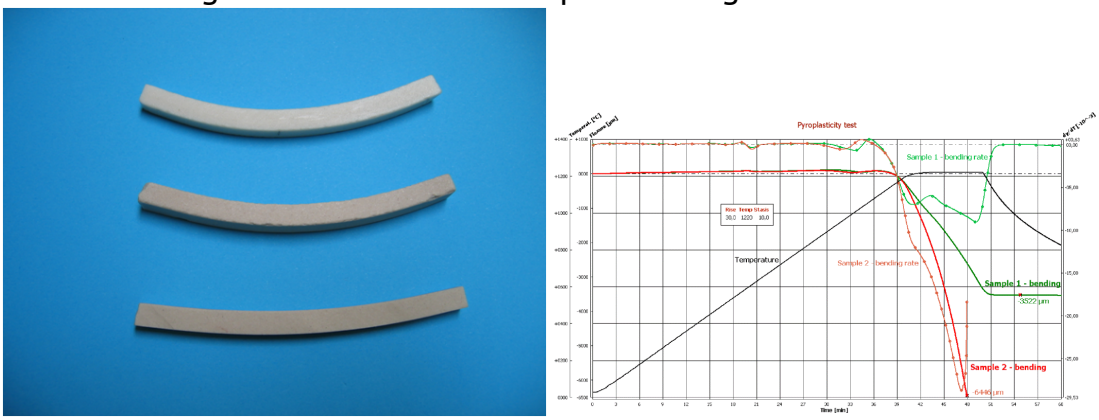


Figure 1.30: Deformation upon wetting during glazing

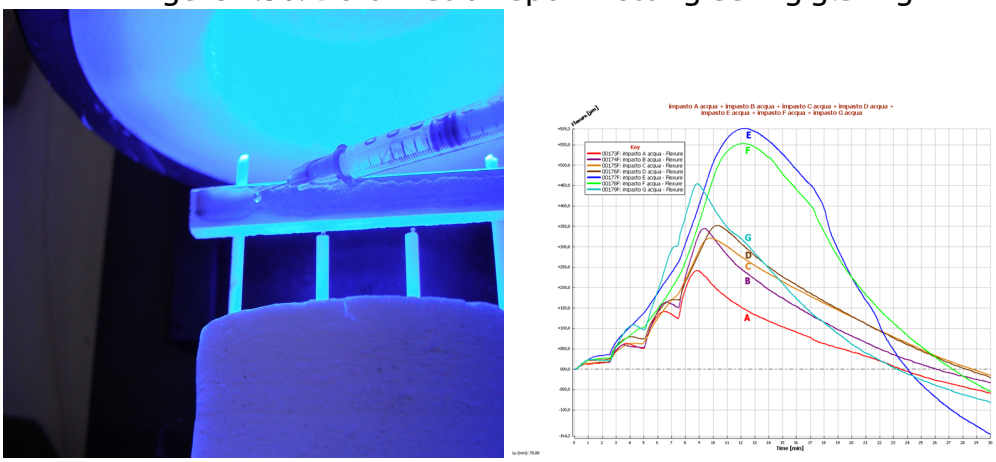
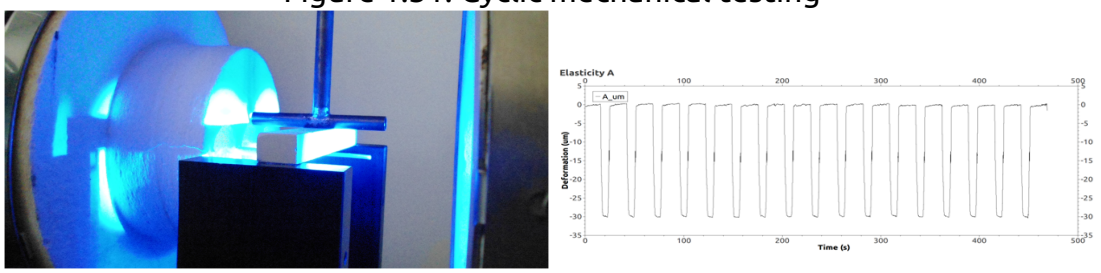


Figure 1.31: Cyclic mechanical testing



of water one side of a green specimen. The behavior of specimen of slightly different composition shows quite big changes in the deformation (Figure 1.30).

A new development to use the same optical system for true mechanical testing, was presented in 2011. The market is full of near to perfect mechanical testing machines and there most probably is no need for another one. The idea behind this project is to offer a plug-in tool to enable the user of a MISURA-FLEX machine to perform a very specific mechanical test which would otherwise require a big investment.

The mechanical testing of traditional ceramic materials does not take in considerations parameters like modulus of elasticity and plastic deformation under load. The measurement of these two parameters on green specimen is of great help in understanding the reason of cracks and failure that may happen on the manufacturing line.

Cycling testing on green samples, can be performed with a modest investment installing a simple plug-in on a MISURA FLEX machine (Figure 1.31).

One more interesting feature of this line of instruments is the fact that several features may be incorporated into the same architecture. Expert System Solutions currently supplies combined instruments, in any combination between Optical dilatometer, Heating Microscope, Fleximeter and DTA.

1.3.1 Optical Fleximeter Applications

This subsection enumerates some prominent applications of the fleximeter instrument, by verbatim quotation of most interesting works.

1.3.1.1 Traditional Ceramics

Generally, the deformations of ceramic materials during and after firing may have a complex origin. If the products are made up of a single material, such deformations are mainly due to pyroplastic phenomena. In the case of multi-layer materials (like glazed ceramics or double charge technology products), two further factors must be considered: the state of tension established between layers, and their differences in behavior during sintering.

The optical contactless experimental technique providing bending data during the heating treatment, is the best available tool to approach this problem. Combining it with optical dilatometry tests, a complete investigation of elastic and plastic deformations occurring in traditional ceramic materials is achievable [Davide Sighinolfi, 2012].

Pyro-plastic deformation is a problem also in the manufacturing of porcelain tiles. Specifically for tile ceramics fired in roller kilns, when the tiles are moving along the kiln carried by the rollers it is possible that a piece can bend to accomplish the roller rotation because it is submitted to vertical forces due its own weight. As a result the tile production is affected by curvatures arising in the final product. The pyroplastic

deformation occurs more frequently in highly vitrified pastes like porcelain tiles. [Bernardin et al., 2006].

Porcelains are vitreous ceramic white-wares used extensively in tableware, sanitaryware, decorative ware, electrical insulators and dental prosthetics. Vitrification indicates a high degree of liquid phase formation on firing which confers low (often <0.5%) porosity and high (>40%) glass content on fired porcelains. Firing of porcelain bodies promotes physico-chemical reactions, responsible for the final properties of ceramic products. In this process, it should be considered the kinetic limitations, the development of the phases, and the complexities of the microstructure. Raw material preparation, drying conditions and firing cycle are going to have a strong influence in the product qualities. Influence of firing regime is related to the kind of furnace, firing atmosphere, maximum temperature and soaking time. All these parameters affect quality and cost of the products [E. Eren and Ay, 2007].

1.3.1.2 Gas Turbine Coatings

Modern gas turbines for power generation and aero-engine application experience continuous efforts towards an increase in thermal efficiency. From the materials point of view particular focus is directed to the gas turbine blades and vanes, especially of the first stage, which must withstand not only mechanical loads, but also the high temperature exposure and oxidation attack.

Due to the good mechanical properties of nickel-based superalloys, e.g. high strength up to 1000°C and resistance against high temperature inelastic deformation, they are widely used for the structural components of the high temperature turbine section. In order to protect the base material of turbine blades against high temperature oxidation and corrosion, two types of metallic coatings are typically applied, which are known to form dense protective oxide scales: Aluminides or Pt-modified.

Aluminide coatings are normally applied by pack cementation and NiCoCrAlY, CoCrAlY, NiCrAlY – layers are coated by vacuum plasma spraying (VPS), low pressure plasma spraying (LPPS) or high velocity oxyfuel spraying (HVOF). The ceramic thermal barrier coatings (TBCs) are applied on top of the oxidation protection layer either by electron beam physical vapour deposition (EB-PVD) technique or by air plasma spraying (APS).[Postolenko, 2008]

The thermal and mechanical stresses arising in the interface between the metal and the thermal barrier are being investigated using the optical fleximeter in combination with the optical dilatometer.

Chapter 2

Software Overview

A new software was developed in order to accomplish the complex task of controlling a modern measurement instrument. The work was based upon existing software by Expert System Solutions: Misura 3. Features and shortcomings were analyzed, and possible solutions were grouped into the Misura 4 development effort.

2.1 About Misura 3

Misura 3 is a desktop application dated 2001, written in Visual Basic® 6 programming language with small image analysis libraries written in C++. It runs in all flavors of Microsoft Windows® starting from Windows XP®. It acts as both control software for performing the actual measurement and main interface for managing, visualizing and elaborating obtained data.

Misura 3 is organized in executable files (modules), which share and reuse big portions of code. The program may be divided into System, Data and Acquisition modules, based on their functionality.

System Modules System modules are used to configure general software and hardware options and to launch other modules.

The Main.exe module is the user gateway to all the functionalities of Misura 3. It offers buttons to access all other modules, plus a settings window from within all software and hardware options may be accessed.

Hardware.exe is a mixed listing of simple hardware and software options which define, for example, communication ports, security features, instrument behavior. It is used as an editing front-end for a text file, Hardware.ini, read by all other modules during initialization.

Tr2416.exe allows to configure the Eurotherm TR2416 Thermoregulator used for temperature control.

Data Modules Data modules provide test data organization and visualization. Archive.exe gives access to the database file where all tests are stored, providing customized listing, filtering and searching of performed tests based on various parameters (test type, name, date). Some metadata, for example the heating cycle

and other test configuration parameters, can be accessed in this way. Images and characteristic shapes are visible for heating microscope tests.

The Gs.exe module (Graph) can plot the acquired data and do some analysis on it (derivatives, expansion coefficients, etc). Multiple tests may be loaded in a multi-Y, single-X plot, where the X axis can be set to Time or Temperature.

PostAnalysis.exe module is identical to Frits.exe acquisition module except that it does not read data from hardware, but from a saved test, performing again the image analysis and shape-detection procedures.

Acquisition Modules The real measurement process is controlled in these modules, one per each instrument kind:

- Frits.exe for the Heating Microscope with single and double samples.
- Frits3.exe for the Heating Microscope with 2 connected cameras, allowing from 1, 2 or 4 simultaneous samples analysis.
- Drop.exe for the Surface Tension calculation using the Sessile Drop method.
- DilatometerV.exe for the vertical Optical Dilatometer instrument.
- DilatometerH.exe for the horizontal Optical Dilatometer instrument.
- Flex.exe for the Fleximeter instrument.

Acquisition Modules are quite similar in their functionality: they show the real-time image acquired from the connected cameras, the temperature, some analysis results. They allow to configure a new test setting description, heating cycle, acquisition intervals and termination conditions for ending the analysis.

2.1.1 Limiting characteristics of Misura3

Misura 3 is a mature application which perfectly meets the requirements of the measurement instrument for which it was developed. Nevertheless, prolonged laboratory use, new installations and customer assistance showed some critical points. Moreover the market urge towards the development of new, highly customized instruments definitely proved the general architecture of Misura 3 as dated. The way it is programmed, the framework and development environment it is based upon, imposed a renovation from the ground up.

Development Ecosystem Visual Basic 6 (VB6) has been a hugely successful development language. Many companies are still maintaining VB6 code [Nelson, 2007], even if its support was dropped by Microsoft in 2005. Nonetheless, finding and integrating up-to-date external libraries is now quite difficult. As a result, many tasks are accomplished in an elemental way.

For example, the thermoregulation is shifted to an external hardware device, which has limited configurability of the temperature curve and basic control algorithms. The image analysis is limited to 640x480 pixel resolution, and uses only few

points and simple mathematical calculations. Small image defect may lead to great analysis errors or scarce reproducibility.

The language does not support many features which modern development paradigms require. The most macroscopic missing is real Object Oriented Programming support [Wikipedia, 2012a], due to the lack of a proper inheritance between objects. Its implementation has limitations which force unnecessary complexity in code organization. For example, each class must be written in a separate file; shared code must reside in modules forming separate files; graphical interface code also is into other files.

As a partial consequence of this insufficient OOP support, large projects become quite impossible to manage.

Evidence comes by doing a quick comparison amongst the 540,171 public code repositories scanned by the Ohloh website dedicated to code metrics [Black Duck Software, 2011] in Table 2.1.

Table 2.1: Ohloh data about programming languages adoption

Programming Language	Scanned Projects	Total Lines	Lines per Project
C	53,321	2,636,786,864	49,451
C++	43,517	1,390,689,234	44,704
Java	65,668	1,230,512,219	18,738
Emacs Lisp	3,308	50,219,362	15,181
C#	22,538	284,642,019	12,629
Python	43,603	254,430,952	5,835
Visual Basic	4,279	21,377,083	4,995

The "Scanned Projects" column is calculated by summing any project in which there is at least one line of code in the corresponding programming language.

The column "Lines of Code per Project" shows projects which contain C or C++ code have big portions written in these languages (statistically, more than 44,000). On the other hand, Visual Basic projects are much smaller (less than 5000 lines).

Part of this statistical differences may be due to "language verbosity", but an entire order of magnitude cannot be explained with this argument. Developers are making a very neat choice depending on the dimension of the application they want to write.

Unmaintainable Source Code The inherent limitations of the development platform certainly contribute to organize code in a maintainable way. Nonetheless, they cannot be accounted as the major contributor to the awkward final result of Misura 3 source code.

The abuse of some code some practices and constructs obstructs detecting the effect of a change, or the cause of a bug:

- Conditional compilations. Some portions of code are compiled, some are ignored, depending on the target executable being created.

- Strong interdependence between shared modules and graphical objects. Shared modules refers to myriad of graphical objects, which must be defined in forms and hidden someway if not used.

The lack of a proper development process may be accounted as the major contributor for bad code organization. Misura 3 does not take advantage of a version control system, automated testing and bug tracking. This makes hard to find, manage and avoid conflicts, bad constructs and bugs.

The alternation of four different project managers in less than ten years did not help cultivating a common development vision and direction, forcing new programmers to find ugly workaround to urgent problems.

User Interferences All acquisition hardware, like cameras, thermoregulators, stepper motor control boards, input/output boards, must be connected to the computer where Misura 3 runs: up to 9 USB ports may be required to control more complex instruments. All related drivers must be carefully installed and tuned. The user interactions happen in the same computational environment where all these delicate conditions must be preserved. In the meanwhile, mission critical and resource consuming acquisition process runs.

This condition frequently leads to conflicts with user-installed software (anti-virus, data elaboration suites, enterprise applications), and cannot guarantee that 100% of system resources are dedicated to the acquisition process, sometimes causing the loss of data points.

Single Process Application Misura 3 is a single process application, not using all the power of modern multiprocessing environments. This leads to strong limitations in the acquisition of data points: each second all connected devices are subsequently read, then results are elaborated and stored. These operations require some time (50-200 milliseconds) blocking the program from any other interaction (also with the user).

This implies that as the number and throughput of connected devices increase, as the complexity of data elaboration flourishes, the program experiences longer delays, slows down and eventually records less points - or completely blocks.

Outsourced Thermoregulation The temperature control is managed by an external, industrial-grade device. While this implies great robustness, the control patterns are limited by those implemented in the device itself. The external device only allows a limited set of configurations, such like control algorithm variables and thermal cycle curve, which are inadequate for advanced scientific applications. For example, only up to 16 setpoint segments are executed in the same cycle run, where each segment consist of a straight line or a dwell. Complex time-temperature functions or feedback interaction between measurement and temperature are not possible beyond start/stop events.

Inadequate Database Platform Data are stored in a simple Microsoft Access database, not adequate for large arrays of numbers generated from scientific instrumentation. The monolithic structure of the database makes it impossible to add data to be recorded without breaking the compatibility with the installed customer base.

Data exchange with other Misura 3 instances is cumbersome and export features versus other data format are limited. Extending these features, although strongly needed, would require a considerable amount of tedious and redundant programming.

Insufficient Configuration Management Most of the program parameters are stored on either in external devices, or hard-coded inside the source code. The insufficient Hardware.exe application, and related Hardware.ini file, cannot afford the complexity of all the parameters needed for complex equipment. One prominent reason is that Hardware.exe/.ini model is limited to parameters which are read-only during the measurement. Some fundamental parameters require read/write access, and are separately stored on the Windows Registry.

For example, camera configurations rely completely on Windows drivers, without storing any parameter. Each time a camera USB connector is moved, and sometimes even spontaneously, all camera settings are lost and must be manually reconfigured. Camera drivers conflicts are quite common and difficult to trace down to a cause, their source code not being public.

2.2 Misura 4 general characteristics

The Misura 4 development starting points were the previously enlisted Misura 3 limitations.

The first architectural defect of Misura 3 was its development ecosystem, which is dated and insufficient for its final use.

Open source development tools and libraries were chosen in order to improve productivity. Misura4 is written in Python, a mature programming language with a large set of extension libraries useful for scientific data processing. Continuous Integration methodology and tools were adopted in order to increase code quality and speed up the work.

A client/server model was implemented, in order to insulate hardware and mission critical code execution (server) from user interferences (client), and decouple their development paths.

The Linux operative system was adopted as the main development platform and as the computing environment for the Server. The Client was developed using cross-platform tools, leaving to the customer the choice between Linux, Windows and Mac OS X.

Parallel processing is a native characteristic: many tasks are spawned amongst available processors, and a powerful shared registry was implemented in order to share parameters, results and states between the processes.

A specifically optimized file format for large numerical data, HDF5, was favored against more general-purpose database systems.

Configuration of devices and algorithms is comprehensive and completely exposed to the user and documented. Hard-coded parameters are avoided, and the few which still survive are all grouped in a single module.

All at once these aspects constitute a revolution, both in the software architecture and in the development process, and are expected to produce an advanced, robust and flexible application.

In the following subsections each of these choices is explained and supported by proper references.

2.2.1 Software Development Ecosystem

According to [Georg von Krogh, 2007, Zymaris, 2009, Ibanez, 2011, Frigeri and Speranza, 2011], the open source phenomenon is the application of the scientific method to the process of software development. Through reproducibility of results and peer review, open source tools are considered as the only eligible ones for doing scientific research.

This similarity between science and open source software development might explain the innovative effect it has on the technology [Bitzer and Schroder, 2005, Reese, 2000, UNU-MERIT, 2006], and the high availability of quality code for scientific computing, networking, database and graphical tasks.

The immediate accessibility of open source tools makes it very simple and inexpensive to evaluate and choose the best solution for any given problem. Access to the executables without any license management issue is a key aspect. Evaluating a proprietary package extensively, without buying it, is frequently impossible. It is even more difficult to accept the idea that an evaluation error was made, and money wasted.

But accessibility extends also to the entire development cycle of the product: mailing lists, user comments, bug tracking, and obviously - the entire source code. Those are precious sources of information about product quality, which can easily be considered to evaluate its implementation.

The availability of source code guarantees that any problem will have a solution, if it is important enough to be resolved: a key requirement for a truly long term project.

The Python Programming Language Python is a general-purpose, high level, multi-paradigm programming language, with code readability as a major target. It has a large and comprehensive standard library (often referred as "batteries included"), and a myriad of extension libraries for scientific computing.

Python is often referred as a rapid-prototyping programming language [Summerfield, 2007, Lutz and Prechelt, 2003], meaning that its ease of use and extendibility facilitates the development of new ideas, that will be re-implemented in the production environment (for example, legacy software) once proved good.

Table 2.2: Google Scholar search results returned for major programming languages.

Year	Python	Visual Basic	C#	Java	Fortran	C++
2011	10900	11900	7880	38700	15900	1980
2010	10900	13100	8800	43500	17200	2050
2009	8990	13100	8110	49300	17100	2140
2008	7910	13200	7720	54100	17300	2330
2007	6450	13900	6790	57500	18000	2390
2006	5170	13800	5690	60200	17200	2270
2005	4130	12900	4490	58600	16400	2240
2004	3120	11900	3470	54400	15700	2100
2003	2410	10800	2450	47300	14400	1740

Table 2.3: ScienceDirect search results returned for major programming languages.

Year	Python	Visual Basic	C#	Java	Fortran
2011	661	385	284	1901	1651
2010	436	394	207	162	1476
2009	319	401	180	1518	1509
2008	229	369	126	1452	1548
2007	239	435	104	1367	1464
2006	177	419	80	1244	1553
2005	152	359	58	998	1482
2004	118	335	53	925	1397
2003	94	212	31	719	1385

The scientific community noticed the usefulness of this language and is increasingly adopting it in many fields. This impression is quite difficult to prove over, but some simple searches performed on two major academic literature databases, Google Scholar [Google, 2012] and ScienceDirect [Elsevier B.V., 2012], support this statement. The hypothesis is that if a programming language is solving cutting edge scientific problems in a given moment, it will be referenced in published works. So, by tracking in time the search results containing references to a programming language, we can observe its adoption or abandon.

The search on Google Scholar was performed asking all published data containing "*name AND software", where *name is the programming language name: for example, "python software". Results are listed in Table 2.2 and plotted in Figure 2.1.

All considered programming languages are experiencing decreasing search results in Google Scholar, starting roughly around 2007-2008. The only relevant exception is Python, which since 2003 had a steady increase.

The ScienceDirect search was restricted to journal articles in these fields: Biochemistry, Genetics and Molecular Biology, Chemical Engineering, Chemistry, Computer Science, Energy, Engineering, Materials Science, Mathematics, Physics and Astronomy. Compared to GoogleScholar, ScienceDirect has a much more limited database, exclusively composed by relevant scientific journals.

The ScienceDirect results show increasing trends in all programming languages,

Figure 2.1: Trends in Google Scholar search results about programming languages.

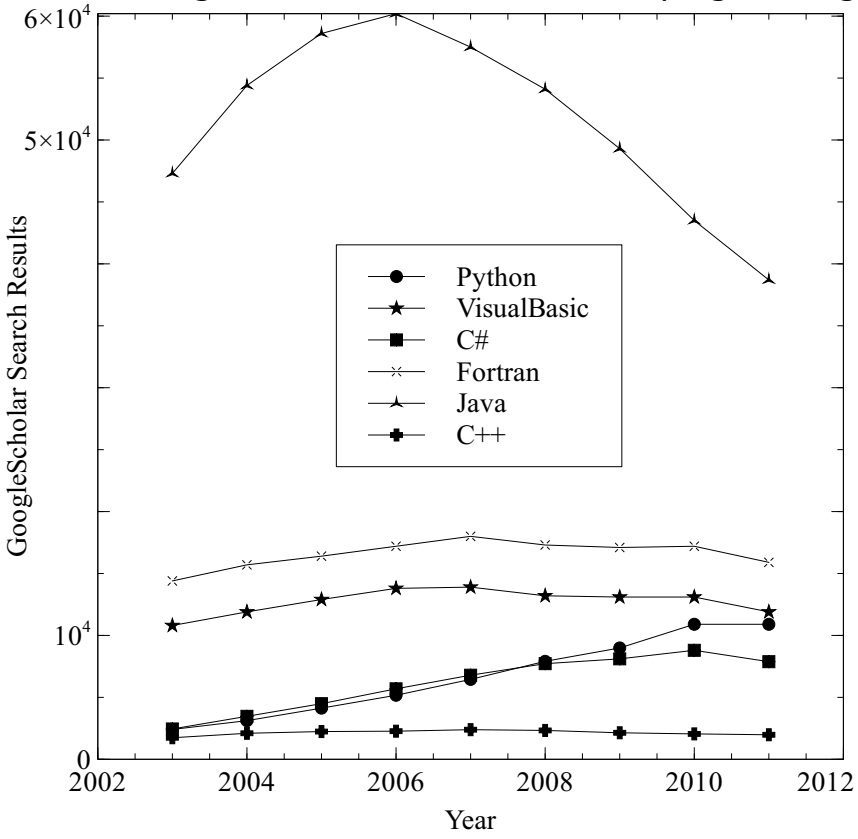
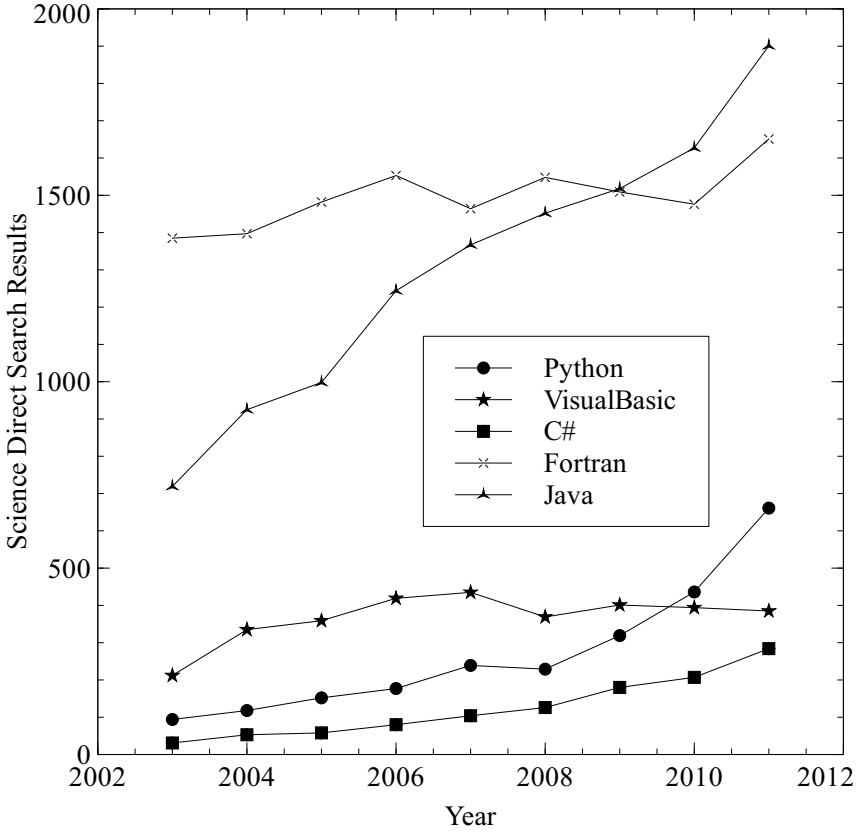


Figure 2.2: Trends in ScienceDirect search results about programming languages.



with Java and Fortran being the most talked about before Python. The latter shows and exponential growth, and in the last three years has the highest increase rate. I assume ScienceDirect data to be more meaningful, being restricted to high quality papers in specific fields.

This simple method does not distinguish between paper about the language per-se (for example improving or documenting it) or the language as a tool used to solve problems. In my opinion, trends are sufficiently clear for a comfortable interpretation.

The Python programming language is gaining the attention of the scientific community in increasing rate. Its open source nature lets us expect that this wider adoption of advanced users will make more cutting edge features and libraries available in the future.

Precedent Table 2.1, about lines of code per projects written in various languages, seems to discourage using Python for big projects. These data clearly showed that Visual Basic was used only for small projects, 5000 against 45000 mean lines of code written in C++.

We must carefully consider a key characteristic of Python: it is widely considered as a scripting or integration language. This means that is very common to find small utility Python script inside huge projects where the main language is another one. Since Ohloh builds the statistics accounting each project with at least one line written in a language into the corresponding category, Python clearly has a small count. On the contrary, VisualBasic is a monolithic development environment, which means that in most of the cases it is the main project language.

Misura 4 actually largely uses Python as an integration language, where most of the non-key features are taken from external open source tools. The technologies empowering Misura 4 are briefly described:

Scientific Python. Scientific Python (SciPy) and Numerical Python (NumPy) have been adopted as mathematical computation libraries. They handle interpolation, fitting, optimization, vectors, matrices. Many Fortran and C libraries are accessed from Python through these two packages (they are ‘‘ wrapped’’). Wrapping means obtaining the speed of low-level libraries (Fortran) into an high-level environment, which is Python.

Open Computer Vision Library. Open Computer Vision (OpenCV [Bradski, 2000]) was born as an Intel Research initiative in 1999. Intel considered it a driver for intensive CPU applications, and seen a first public release in 2000. Since then, the community of users and developers largely moved outside of Intel laboratories, and is now an independent project with various individual and corporate contributors.

It has a versatile Python interface but the core is written in C++ with optimized C for computation intensive tasks. The computing load may be distributed over multicore processors and, recently, modern Graphics Processing Units (GPU) implementing Nvidia CUDA architecture[Nvidia Corporation, 2007].

With more than 500 functions and 2500 optimized algorithms, it spans many areas in vision, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision and robotics [Bradski and Kaehler, 2008]. A Machine Learning Library is also included for statistical pattern recognition and clustering.

Misura 4 takes advantage of OpenCV features for part of the image analysis, especially image preprocessing and filtering, features and pattern recognition, geometric description. Once the shape of the sample shadow is calculated from the acquired image, Misura 4 own algorithms take over.

Twisted Networking Engine. Twisted is a pure Python event-driven networking engine for developing Internet applications. It simplifies the task of implementing custom network client and server applications, and supports many common network protocols, including TCP, SMTP, POP3, IMAP, SSHv2, HTTP, XMLRPC, SOAP. Among preeminent Twisted users there are NASA, Lucasfilm and Canonical.

Communication between Misura4 client and server applications is managed through the Twisted library. The main Misura4 server process is an instance of a Twisted TCP server class.

Hierarchical Dataset Format. The file format for storing huge amount of raw data during acquisition is the Hierarchical Dataset Format[The HDF Group, 2010], through the wrapping library PyTables[Alted et al., 02]. HDF is proposed as an efficient way of storing and accessing large, complex, heterogeneous data with parallel and random input/output. Example fields of application of HDF are astrophysics, geography, high energy physics. Between excellent users, they list NASA Earth Observing System, The Atmospheric and Oceanic group at NCSA, General Atomic Fusion Group at Princeton, Global Change Research Program at U.S. Geological Survey, NeXus neutron and X-ray scattering at Argonne National Laboratory, and many other.

Qt Graphical Toolkit. Python has language bindings towards a variety of graphical toolkits: TK, GTK, Microsoft Foundation Classes (MFC), wxWindows, Java/SWING/SWT, .NET, FLTK, OpenGL.

Qt (by Nokia) is a mature, comprehensive cross-platform C++ toolkit not limited to Graphical User Interfaces, such has command line-tools, consoles and servers. The combination of Python and Qt (PyQt) makes it possible to deploy applications on Linux, Windows, MacOS X, most Unix-based systems and many embedded devices[Summerfield, 2007].

The portability was fundamental in choosing Qt, as the development is completely done on the Linux platform while most users will need Windows/MacOSX GUIs. Its uncommon capabilities, including networking, database and multithreading, were welcomed in a client-server application dealing with big data.

Algorithm 2.1 OpenCV Memory Leak bug report snippet

```
1 from numpy import random, array
2 import cv
3 x=random.rand(50000)
4 y=random.rand(50000)
5 seq=array([x, y]).transpose()
6 for i in range(100):
7     cv.Moments(seq)
```

Veusz Scientific Plotting Package. Veusz (pronounced as "views") is a complete scientific plotting and graphing application written in Python, PyQt and NumPy, designed to produce publication-quality plots [Sanders, 2012]. Plots are created by building up plotting widgets with a consistent object-based interface.

The program provides its own scripting interfaces: its native file format is a valid python script. It is also easy to extend it using a plugin mechanism, for importing new data formats or doing arbitrarily complex operations with datasets and graphical elements.

The clever Veusz object-oriented architecture allows easy integration into Misura4, by using it "as a library" which contributes graphing capabilities or by extending menus and widgets with Misura4-related objects.

GNU/Linux Operative System The GNU/Linux operative system had high consideration for networking tasks since its very beginning, and is now well established [Linux Foundation, 2010]. The Linux kernel has been successfully deployed also in mobile devices, under the Google Android project, knowing a quick diffusion worldwide (45-55% of the market share)[Wikipedia, 2011a]. It also powers most (91%) of the 500 fastest supercomputers in the world, proving to be a robust platform for high performance computing tasks[TOP500.org, 2011]. Many embedded devices, at an increasing rate [VDC Research, 2011], run Linux-based operating systems, taking advantage of its multiprocessing and real-time features.

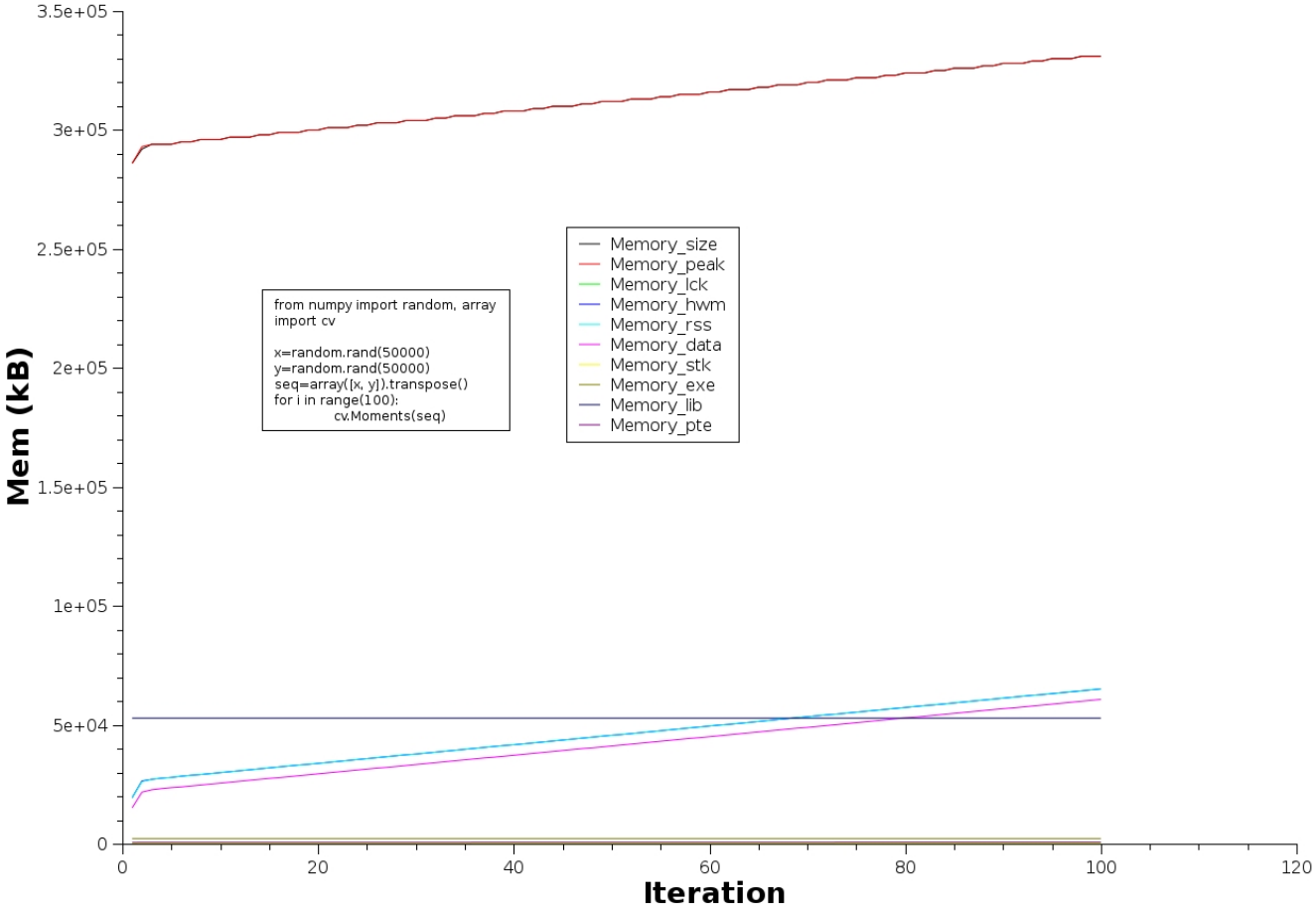
Beside those trends, a GNU/Linux distribution was chosen - also as a development environment - for the advanced packaging systems available, a clean build system and development pipeline, and the possibility to easily create customized installers OEM-like.

2.2.1.1 Story of A Bug

This short paragraph exemplifies the benefits of developing in an open source ecosystem. A memory leak causing the process to use increasing amounts of memory, blocking the computer, was found during a Misura 4 test.

This bug only presented when Misura4 was analyzing images. After a debug phase, a function call from the OpenCV library was isolated. Specifically, it was a call to the `Moments()` function, as exemplified in Algorithm 2.1.

Figure 2.3: Memory size of Algorithm 2.1



The execution of that simple loop, were the Moments of a random sequence of the same `(x,y)` points is calculated at each iteration, caused the memory usage graph in Figure 2.3.

This behavior cannot be explained by the Python code snippet.

I immediately submitted a bug report to the OpenCV Issue Tracker [Paganelli, 2010], explaining the problem and supporting it with proper data.

Few days later I had time to work again on that bug. Getting familiar with a big project as OpenCV might seem too difficult for just one little bug, but the clean development pipeline of Linux distributions (Ubuntu in my case) made it very quick to get and compile the code. Two commands were enough to become productive:

```
apt-get build-dep opencv
svn co https://code.ros.org/svn/opencv/trunk/opencv
```

I eventually found the bug in the OpenCV-Python wrapping code, which provided the interface between the base C function and the high-level Python language. A structure was created, filled with data, used to compute the moments but never deleted: this was causing the leak.

Once I found the problem, I wrote a simple "quick-and-dirty" fix in order to resolve it, then I created a diff file and sent back to the issue tracker (Algorithm 2.2).

Algorithm 2.2 Quick fix for the cv.Moments memory leak

```
Index: gen.py
===================================================================
--- gen.py (revisione 3947)
+++ gen.py (copia locale)
@@ -261,6 +261,8 @@
          if len(joinwith) > 0:
              yield '  return shareData(pyobj_%s, %s, %s);' \
                                  % (joinwith[0], joinwith[0], all_returns[0])
          else:
+             if cname(name)=='Moments':
+                 yield '  cvReleaseData(arr.v);'
              yield '  return FROM_%s(%s);' \
                                  % (safename(typed[all_returns[0]]), all_returns[0])
        else:
          yield '  return Py_BuildValue("%s", %s);' \
                        % ("N" * len(all_returns), ", ".join(
                                  ["FROM_%s(%s)" % (safename(typed[n]), n) for n in all_returns]))
```

Anyway, this was not the definitive solution to the problem. I just added a special case to a function, asking a special attention while generating the wrappers for the `'Moments'` name. From the superficial analysis I did while resolving that bug, I found many other situations where memory could easily leak. I reported that other functions may be affected, and I only resolved the problem for my case.

Two months later my tiny patch implications were fully understood by the OpenCV development team, which found a general solution for the problem and avoided future memory leaks by integrating an extensive test suite.

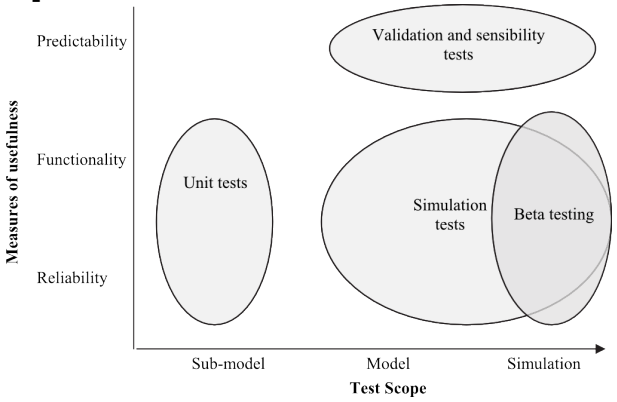
A similar problem in a commercial closed-source third-party product could produce a big damage to a company. The application-side workaround may not be possible, or may require much more effort than resolving the library-side original problem. If the vendor proved to be too slow to patch small issues like that, a custom Moments function had to be written (with no expertise at all) or the feature removed from Misura 4 (not quite possible).

2.3 Development method

From the start of Misura 4 software development, a considerable effort was invested in adhering to the Continuous Integration method (CI). CI implements continuous processes of applying quality control. By performing frequent, small, and highly automated quality assessment tests, it is possible to improve quality of software in less time [Saff and Ernst, 2004, Boshernitsan et al., 2006, Brun et al., 2011, Holzworth et al., 2011, Bertolino, 2007, Osterweil and Clarke, 1990, Torkar, 2005].

For example, Staff D. and Ernst M. published in 2004 an Experimental Evaluation of Continuous Testing During Development. They performed a controlled human experiment over a group of students, comparing productivity, quality and satisfaction obtained with or without CI. They measured an improvement of two or three-fold in speed and success rate of students using CI techniques.

Figure 2.4: Scope and Usefulness of Testing Practices, according to [Holzworth et al., 2011]



2.3.1 Version Control System

The first step towards Continuous Integration in Misura4 development was the adoption of a Version Control System which records changes in the source code, documentation or configuration files. Every time a change is made to any part of the software, and can be considered finished, the developer "commits" new updated version to a central repository, attaching a short note to describe it in human language. The repository automatically assigns a progressive version number and records all the differences towards the latest recorded version.

Then it is possible to go back in time: viewing the differences between any revision, merging modifications made by colleagues and reverting to any older revision. This chronological record proved to be extremely important to resolve regression bugs and version conflicts amongst collaborators [Brun et al., 2011].

The version control tool introduced was SubVersion [Apache Software Foundation, 2012], an enterprise-class centralized repository. It was founded in 2000 by CollabNet as an open source project, and is now developed by the Apache Software Foundation. It is widely adopted in both open source and corporate world.

2.3.2 Automated Testing

Another strong requirement for Continuous Integration is automated testing, which includes Unit, Simulation and Validation Testing. All of them consist of writing a program which checks another program - at increasing levels of complexity.

Holzworth et al. [Holzworth et al., 2011] explain how these testing practices cover three different scopes and measures of usefulness (Figure 2.4). The scopes refer to the target of modeling a natural phenomenon through software. The author's thesis is that by implementing Contiuous Integration methods, also model quality increases. We can generalize Holzworth graph by translating Sub-Model as function, method or algorithm (Unit); Model as module or significant part of the software; Simulation as complete software behavior and interaction with real data.

The Python standard library offers a complete testing framework, the `unittest`

Algorithm 2.3 Testing the append method from the data.CircularBuffer class.

```
1  import unittest
2  import numpy as np
3  from misura4 import data
4
5  class CircularBufferTest(unittest.TestCase):
6          N=50
7          def setUp(self):
8                  d=np.arange(N)
9                  t=np.arange(N)/2.
10                 self.lst=np.array([t, d]).transpose()
11                 self.cb=data.CircularBuffer(N)
12
13         def testAppend(self):
14                 for t, d in self.lst:
15                         self.cb.append([t, d])
16                 self.assertEqual(len(self.cb), self.N)
17
18 if __name__ == "__main__":
19     unittest.main()
```

module [Python.org, 2012], which supports test automation, sharing of setup and shutdown code, aggregation of tests into collections, and independence of the tests from the reporting framework.

Its very general design does not limit `unittest` to "Unit" type of tests, despite its name, but is conveniently exploited for any testing task.

The `unittest` module provides classes that make it easy to support these qualities for a set of tests.

To achieve this, `unittest` implements the following important concepts .

Test Fixture A test fixture represents the preparation needed to perform one or more tests, and any associated cleanup action. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.

Test Case A test case is the smallest unit of testing. It checks for a specific response to a particular set of inputs. `unittest` provides a base class, `TestCase`, which may be used to create new test cases.

Test Suite A test suite is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.

Test Runner A test runner is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.

The example Algorithm 2.3 is a portion of a test suite for Misura4 `data` module, where fundamental data and configuration management classes are defined. The `CircularBufferTest` class is an instance of the `unittest.TestCase` general class.

The `CircularBufferTest.setUp` method is the test fixture, where sample data and target class is created: a list of points [(t0,d0),(t1,d1),(t2,d3)...(tN,dN0)] and a empty `data.CircularBuffer` instance of length N.

Then the `CircularBufferTest.testAppend` method tries to append each point contained in the sample list to the target `data.CircularBuffer` instance, using its `data.CircularBuffer.append` method.

Then the `assertEqual` method, inherited from the `unittest.TestCase` class, verifies that the final length of the buffer corresponds to the number of appended points. If this assertion is false, this test fails and the failure is reported at the end of the complete test suite dedicated to Misura 4 fundamental data types, run after each commit to the version control repository.

Unit Testing Unit Testing refers to the smallest working piece of software, a unit. Usually a unit corresponds to a function, or to the method of a class. A predetermined input is given to the unit for processing, then the output is compared with theoretical output data: if theory and practice differ, the unit test fails and the developer is alerted.

Each test case should be independent from the other, so that the output can be uniquely traced back to a small piece of code.

The main advantage of unit testing is to force decoupling of functions, since single functions are much easier to test than many coupled functions. This increases the possibility to re-use code, and diminishes the effort needed to find the cause of a bug. Unit tests also assure that source code refactoring will not interfere with existing and needed functionalities.

Thus, Functionality and Reliability are the main outputs, in terms of usefulness, of Unit Testing activity.

Validation and Sensibility Tests Validation tests check the correspondence of software behavior towards requirements or samples of real data. Frequently they require a bigger portion of code to be tested, in order to obtain the level of functionality needed to deal with a real, complex problem.

For example, in Algorithm 2.4 a theoretical sample shape (Figure 2.5) is created then fed into the analysis routines. An entire part of Misura 4 Server needs to be instantiated in order to run a complete analysis for a single frame, as shown in the `HSMAnalysis` class definition.

After the analysis on the ideal sample, the obtained values are compared to the theoretical ones: if the difference between the two is more than tolerance (10^{-11}), the test fails. This validation permitted to find and correct bugs which otherwise would be almost impossible to be noticed by the developer. Upper sample angles and width calculation, for example, were found inaccurate due to a wrong configuration option.

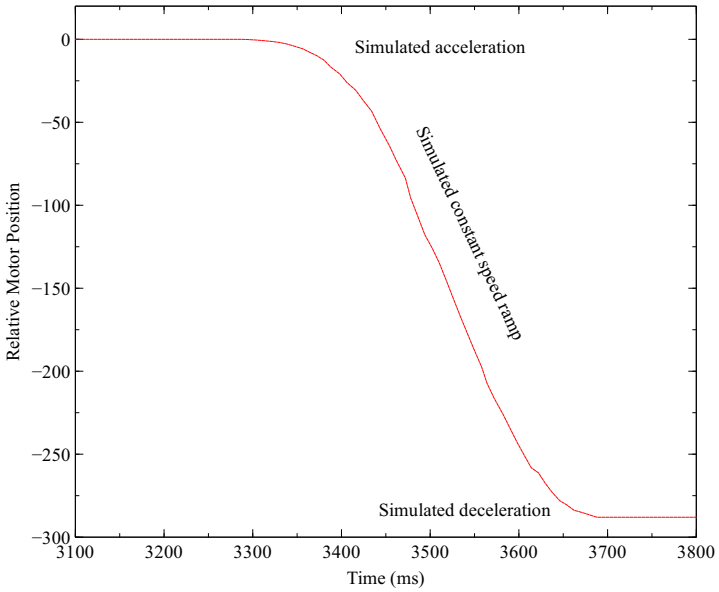
Simulation Tests Simulation Tests take care of the interaction of many parts of the software, up to a complete acquisition process. Hardware behavior is simulated by

Algorithm 2.4 Validation test for HSM analysis

```
1  import unittest
2  from misura4 import interfaces
3  from misura4 import beholder
4  from misura4 import simulator
5  from simulator import camera
6  from misura4 import analyzer
7  import test_utils
8  import cv
9  tolerance=10**-11
10 # Perfect sample profile
11 hsm=[[(0, 320), (640, 320), (640, 480), (0, 480)],              # Base
12       [(212, 160), (426, 160), (426, 320), (212, 320)]]  # Campione
13 # Theorical values and tolerance requested
14 hsmSmp={'w':[214, tolerance],
15                 'h':[160, tolerance],
16                 'A':[34240, tolerance],
17                 'angL':[90, tolerance],
18                 'angR':[90, tolerance],
19                 'angB':[270, tolerance],
20                 'angC':[270, tolerance],
21                 'angle':[0, tolerance]}
22 #Area tolerance
23 hsmSmp['A'][1]=hsmSmp['h'][0]*hsmSmp['w'][1]+hsmSmp['h'][1]*hsmSmp['w'][0]
24
25 class HSMAnalysis(unittest.TestCase):
26         """Checks HSM analysis"""
27         server=test_utils.dummyServer()
28         vo=camera.VideoObject(server)
29         cam=beholder.beholder.DummyCamera(server=server)
30         cam['Analysis_umpx']=1.
31         cf=analyzer.AnalyzerConfiguration('hsm')
32         an=analyzer.AnalyzerSample(cf, cam)
33
34         def testHsm(self):
35                 """Validate HSM sensibility"""
36                 self.vo['path']=hsm
37                 img=self.vo.get_image(cvformat=True)
38                 sample={}
39                 self.an.reset()
40                 img, status, profile=self.an.analyze(image=img, roi=False, sample=sample)
41                 self.assertTrue(status, msg="HSM analysis failed.")
42                 err=0
43                 msg='Analysis errors:\n'
44                 for key, (teor, tol) in hsmSmp.iteritems():
45                         r=abs(sample[key]-teor)
46                         if r>tol:
47                                 err+=1
48                                 msg+='  /!\ %s: real:%E > tol:%E\n' % (key, r, tol)
49                 self.assertTrue(err==0, msg=msg)
50
51 if __name__ == "__main__":
52     unittest.main()
```

Figure 2.5: Theoretical HSM sample input for validation testing

Figure 2.6: Simulation of motor acceleration/constant speed/deceleration phases during a stepper-board controlled movement



"fake" device control objects, according to physical models or device specifications.

Simulators were written in order to generate plausible data. For example, the motor movement generated by stepper-motor control boards shows configurable acceleration, deceleration and speed values. A motor simulator was written in order to reproduce this behavior during simulation tests of complete instruments.

The simulator receive orders to change position from the main acquisition process. Then starts a separate thread where the simulated position of the motor is changed according to accel/decel ramps. Anytime the acquisition process asks for the current motor position, a realistic value is returned by the motor simulator, as shown in Figure 2.6.

2.3.3 Bug Tracking and Project Management

Misura 4 is a replacement for all Misura 3 features, handled in an optimized way and extended. A project management system allows the developer to organize the work, by enumerating tasks, dividing them into groups and milestones, and giving priorities.

As internal laboratory tests begin, many defects of varying gravity are found, and numerous enhancement ideas and requests are proposed by technician and engineers. A bug tracking system helps prioritizing both bug reports and development tasks, and manage the work in order to obtain most important results in the shortest time.

Trac Integrated SCM and Project Management [Edgewall, 2011] web-based system was chosen for the tight integration of project management, bug tracking, version control and wiki content management. Trac interconnects each of these tools, increasing their value to the developers and users [Uhlig et al., 2011].

For example, a Subversion code commit can automatically close a bug reported in Trac, creating a reference from the commit log to the closed bug and vice versa. This makes much more easy to read the history and the current status of the project.

2.3.4 Modeling

Software modeling started later in the development cycle of Misura 4, mainly as a mean to facilitate communication to co-workers and for presentation in this dissertation. The Unified Modeling Language [OMG, 2011, Roff, 2003], UML, was used to explain use cases, class inheritance and execution sequence of the most complex aspects in the software. UML diagrams were built using Papyrus [CEA LIST, 2012] and the Eclipse Modeling Platform[Eclipse, 2012].

2.4 The Client/Server Model

Misura 4 was designed as a client/server application, where tasks are distributed between at least two independent processes: the server, Misura Server, and the client, Misura Client.

The client/server model allows a total, even physical insulation between the user computing environment and the automatic instrument control environment. The server is designed to be installed on an embedded computer inside the apparatus, hidden from the user view, where all control devices are connected. This allows total control over installed components on the server and greater security: the risk of a conflict between user actions and instrument control is minimized.

Client and Server communicates through the network. A single Ethernet cable is sufficient to connect the embedded Server device to a commodity PC where the Client software is installed. Networking of multiple instruments in a lab is as simple as connecting them to a LAN, Local Area Network. It is even possible to connect a Server to the Internet and remotely control an instrument from anywhere.

As laboratories get bigger, with many instruments producing tens or hundreds of reports at the same time, the advanced user needs to review and elaborate all of them from a single location. A recent study [Ricerche e Studi S.p.A., 2011]observed as international delocalization of Research & Development offices is a growing trend in multinational companies and already quite widely applied. The laboratory often becomes a complex infrastructure with equipment sparse over many production and R&D offices in foreign countries. The distributed nature of the client/server model meets the interconnection needs of this scenario.

The communication between Client and Server happens through a standard protocol called eXtensible Markup Language - Remote Procedure Call (XML-RPC)[Phillip Merrick, 2006]. XML-RPC lets the client call remote procedures (functions) on the server, encoding requests and responses using the XML markup language. The use of a publicly documented standard protocol was preferred in opposition to a proprietary in-house protocol to leave open the possibility of new client interfaces. XML-RPC is widely used for dynamic web services, and supported by many programming languages [Wikipedia, 2011b], so in the future we may write a web-based client or leave to other developers or companies the freedom to create their own client to address specific needs.

Client and server does not share any code. This allows to split development effort, and people, without ever producing a conflict until the procedure call specifications remain constant. For example on the Server side new instruments or new control features will be implemented. On the Client side, the focus will be user-friendly visualization, elaboration and storage, unique client customizations, etc.

This positive splitting of development paths is well evident in the most adopted client/server implementation: the World Wide Web [Wikipedia, 2012b]. Web servers and Web browsers are continuously and independently updated, without the server developers needing to share any information with browser teams except the communication standards set by an international standards organization, W3C [Wikipedia, 2012c].

The tasks are partitioned according to the general scheme described in the next subsections.

2.5 Server Tasks

Misura Server performs any acquisition- or equipment-related task. The server is committed to acquiring and elaborating as much data as possible, during the measurement and even before and after.

2.5.1 Connecting Devices

The Server process runs on the computer controlling the instrument. The first task the server must accomplish is to connect all the devices which make the measurement possible:

- Cameras for image acquisition. Each instrument has from 1 to 3 connected cameras, and they can all work at the same time.
- Thermoregulators or thermocouple readers, for temperature reading and power output. Up to two thermoregulators may be connected and working at the same time.
- Stepper motor control boards. On dilatometers and fleximeters the cameras, along with their optical system, are motorized in order to follow the samples. One or two stepper boards may be used to control the motors.
- Input/Output boards. Some special instruments use digital I/O to control some events, like switching.
- A balance to perform simultaneous weight recording during dilatometry tests.

The first connection to a device is issued by the operative system level, using the native driver included in the Linux kernel. When Misura Server process starts, the `/dev` filesystem is scanned in order to find supported *device files*. A device file, for example `/dev/video0`, is an interface by which any user-space application may interact with the device driver, for example reading and sending data to the connected device.

After a known device file is found, Misura sends some control data in order to exactly identify the device to which it is connected. For example, an USB Serial Port device file (`/dev/ttyUSB0`) may be used to communicate to a wide range of devices: thermoregulators, stepper-boards, I/O boards, balances, etc. All these devices share the same driver, so the same kind of device file is created. So if the device replies to a weighting request, Misura knows for sure it is a balance.

Once the device class has been exactly found, an unique identifier is created using the device serial number if available, or the USB connection path. Until the serial number or the USB connection path remains constant, Misura correctly associates a device to its saved configurations.

2.5.2 Network Publishing and User Access Control

The server announces its presence in the network with a Zeroconf/Avahi/Apple Bonjour mechanism, the same used by many networked devices like printers and NAS. In this way any client can find all Misura Servers running on the same network, without the need of knowing IP or MAC addresses beforehand. During the Zeroconf announcement, the Server sends some basic information about itself, like serial number and supported acquisition function.

Once the client has found the server it wants to connect to, it negotiates a Secure Socket Layer (SSL) connection, where user credentials and all traffic data are protected by industry strength double-key cryptographic AES algorithm.

Once the user is logged in, the server assigns read and write authorization levels to all its requests, and passes them to the functions called by XMLRPC. A published function may evaluate this authorization level and refuse to process the request as unauthorized.

The authorization level is automatically handled is all get()/set() requests towards configuration options, in order to forbid reading or editing of sensible data, as explained in paragraph "Global Configuration". Many other functions, like start_acquisition(), require specific authorization levels in order to be executed.

2.5.3 Global Configuration

Misura 4 configuration system is thought to be capillary and comprehensive. Every parameter which could be sensible for the equipment is managed, so that the fewest parameters are hard-coded into the source code and the whole software gains flexibility.

Each configuration option contains also some metadata, along with the current value:

- a short description, showed in the GUI for better understanding
- the factory default value, for resetting the object to a working state
- the range of admissible values (min/max, list selection, etc), which ensures that submitted data is correct and are used to build graphical control widgets
- the data type (Integer, String, Number, Dict, Boolean, etc), also used for validation and GUI building
- a list of special attributes (ReadOnly, Runtime, History, Hard, etc), which instructs the server about how to manage the option
- read and write authorization levels, which hide some options to non-authorized or uninterested users. Authorization avoids both the confusion which may arise from more than 1300 values Misura 4 stores and protects sensible configurations.

Configuration options are stored on encoded text files (.csv) and saved in a meaningful file-system hierarchy. Text files were chosen in this development phase against database platform, preferring total transparency during the debugging process to performance at runtime. These files really require performance only during the startup of the server, when they all are read in sequence (some of them, multiple times). The simple encoding permits great portability towards other applications (spreadsheets), and easy human reading.

No external memory is trusted by Misura 4. For each device, Misura 4 creates and stores an internal copy of all its options. This way configuration losses due to device failure are minimized.

2.5.4 Parallel Acquisition

The Misura server is able to dynamically allocate processes at the start of the acquisition process, where almost all the computational power must be used for maximizing acquisition speed.

Outside of acquisition, the Server is ran inside two processes. The Main Process is the Twisted asynchronous event-loop, for handling clients requests and doing internal periodic tasks. A child Sharing Process handles interprocess communication between the Main Process and any further child process that will be instantiated.

When the acquisition starts, many child processes are created in order to take advantage of all the available CPUs and grant independent scheduling to non-related tasks.

Communicating with hardware, sending and getting data, is the most time-consuming task, especially for high-throughput devices like megapixel cameras. If each device has to be contacted sequentially, then the communication latency of each device in the system would sum up, making data very sparse in time. In this way Misura3 actually could barely get 1 point/second.

In the initialization procedure of the acquisition, each hardware device gets its own process where it reads and elaborates all its data. Output and state values are recorded into short chronological arrays in the Sharing Process.

Another demanding task is collecting all data coming from each device, computing derived values and metadata, managing the overall acquisition, and saving any sensible data to the output file. This is accomplished by a separate Supervisor Process.

This parallel design enables the maximum acquisition speed allowed by each device, and can easily scale with the number of connected devices up to the computational limit of the CPU.

2.5.5 Thermoregulation

Temperature control is entirely operated by the server, inside a dedicated process.

The furnace and samples temperatures are measured with a multichannel instrument, and used by the algorithm in order to calculate the required output power. The power is communicated using a simple DAC interface.

The base control algorithm follows the PID theory, by supplying Proportional, Integral and Derivative factors. This is the same way external thermoregulation devices operate.

The PID algorithm was extended by including physical information about the furnace, like Dissipative and Capacitive constants, allowing greater responsiveness and operation security.

The new algorithm will accept and try to realize any continuous Temperature(Time) function. Moreover, complex interaction between measurement data and temperature may be realized by making the setpoint temperature dependent from another measured value. Rate Controlled Sintering is an example of such kind of interaction: the temperature is regulated in order to keep the sintering rate constant.

See Chapter 6.

2.5.6 Image Analysis

The availability of OpenCV advanced, high performance and open-source image analysis library, opened new possibilities in image filtering, preprocessing and shape detection. The sample is now identified with great flexibility and robustness, being it a cylinder of pressed powder, a drop of molten glass or the border of a dilatometry sample.

A complex set of border analysis procedures was then developed in order to characterize the properties of shapes detected on each frame.

The image analysis is entirely performed in the Server, which also serves frames and analysis results to the Client in a prioritized way. This ensures that frames (as all other data) are served to the Client for visualization only if there are spare CPU cycles remaining after acquisition and analysis.

See Chapter 5.

2.5.7 Data Elaboration

Thanks to the parallel processing feature of Misura4 architecture, the acquisition is performed at the maximum speed allowed by each connected device and by the CPU power. This generates a huge amount of data, which is elaborated both in each device process and in a Supervisor Process. The Supervisor Process also writes the results to the test file, which follows the HDF5 standard file format, optimized for huge arrays and sets of scientific data.

2.6 Client Tasks

Misura Client allows the interaction between the User and the equipment, mainly by providing a Graphical User Interface (GUI) accessing test configuration, data visualization, post-elaboration and management. Server execution is totally independent from Client execution. The Client can disconnect or shutdown in any moment without any server status modification its or activity interruption. For example, if an analysis is being executed, the Client can connect and disconnect without the analysis being affected by Client status. Concurrent Clients are also allowed.

2.6.1 Networking

The Client is able to automatically find any Misura Server instance running on the same network it is executed on, and presents the User the choice about which machine to connect to. The available Servers list is constantly kept updated.

The Client is also able to connect to a special Expert System Solution Simulation Server, for evaluation purpose.

After the machine has been chosen or the address manually typed, the Client handles the SSL connection and login procedure, by presenting a username/password window.

2.6.2 Configuration

The Client allows the configuration of any option present on any Server object, if the required authorization level is obtained. Options are displayed in a visual and interactive way, for example by means of sliders, spin boxes, menus. All graphical elements (widgets) are "connected" to the Server options they represent, meaning that any Client change has immediate effect on the Server and vice-versa. This interactivity improves configuration speed, as configuration changes are reflected on the instrument in real-time and the operator can quickly prove their correctness.

Each option widget automatically displays a documentation tooltip (label) and also a link to an extensive documentation page on the Server.

2.6.3 Acquisition

An unique Acquisition interface is defined for all possible instruments: Kiln, Microscope, Vertical Dilatometer, Horizontal Dilatometer, Fleximeter, Post-Analysis, Drop Analysis. After having connected to a specific machine (see Networking), the user is presented with a list of available measurement functions (useful for combined instruments).

The Acquisition Interface resembles in many parts the Configuration Interface, displaying all measurement, sample and device options with the same graphical elements used by the latter.

In addition to configuration panels, the Acquisition Interface also displays the following widgets:

- A log window, showing remote messages emitted by the Server.
- A table holding all past analysis results.
- Camera windows, one for each configured camera, showing the currently acquired frame and letting the user graphically manipulate some image analysis options.
- A data plot window, showing the acquired curves by time.
- A thermal cycle builder, displaying a graph of requested cycle and a table with configured points.

The Post-Analysis interface defines three more widgets, used to retrieve the data to be post-analyzed:

- A database window, for browsing available Misura4 files.
- A Misura3 database window, for opening the legacy Misura3 files.
- A progress window, showing the post-analysis progress and letting the user pause it on a certain frame.

2.6.4 Plotting Facilities

Advanced plotting is accomplished by extending the functionalities of an existing package, Veusz [Sanders, 2012]. Veusz is a complete scientific plotting solution entirely written in Python. The following specific functionalities were easily added via Veusz's own plugin system, or by subclassing part of its classes:

- Connection to a Misura server and access to its database.
- Misura 4 file format opening, loading and default graph display.
- Opening of legacy Misura3 database format.
- Highly customizable test reports.
- Universal file container.
- Mathematical functions needed for thermal analysis data elaboration.

2.6.5 Database

Misura 4 database is intended as a "test file index". The user puts its Misura 4 test files in a folder hierarchy made as he prefers, then a batch process recursively scans the folder in order to find all test files (discarding any other file). For each file, metadata is read and memorized in an index, so that the user can quickly find a test whenever it is saved in its hierarchy. Graphical elements are provided in order to query the index, preview a file's metadata, or open it in the plotting package.

2.6.6 Scripting

Server function are available not only through the graphical interface, but also using a scripting interface. The Python language can be directly used to write custom scripts which automate non-standard tasks. For example, a custom test termination condition may be quickly implemented client-side by checking a certain condition (eg: expansion is more than 2% and expansion rate is less than 0.01%/°C), an calling stop_acquisition() if verified.

A small module, m4script, is provided in order to facilitate repetitive networking tasks, like SSL connection and login.

2.6.7 Translation

The default language of Misura Server and Client is English. A dynamic translation mechanism has been implemented Client-side in order to translate any visible string into the User language, taking advantage of the standard Qt translation tool-chain. An automation was developed in order to catch and record any non-translated string, facilitating translator work.

Chapter 3

The Server Architecture

This chapter explains how Misura Server tasks are implemented in the software, at the highest possible level of abstraction.

Figure 3.1 shows the UML component diagram for all the modules and packages which constitute Misura Server.

Fundamental modules contain basic services, data structures and utilities accessed everywhere across the application.

3.1 Understanding Parallel Processing and Network Publishing

These topics strongly influence the entire software architecture, so they must be well understood before exploring other aspects.

Parallel processing is provided via the standard Python library, thanks to the module [Python Software Foundation,]. Thus its implementation is very simple, in terms of code lines, but has a huge impact on almost any other single line of code.

The Python `multiprocessing` module is able to spawn multiple subprocesses. Any function can be called into a subprocess, by simply writing few lines (see Algorithm 3.1).

Function execution happens in a separate process, using its own memory space which is initially a clone of the main process memory. So, if a variable changes on the main process, the subprocess is not aware of that change.

The `multiprocessing` module offers some facilities in order to share data

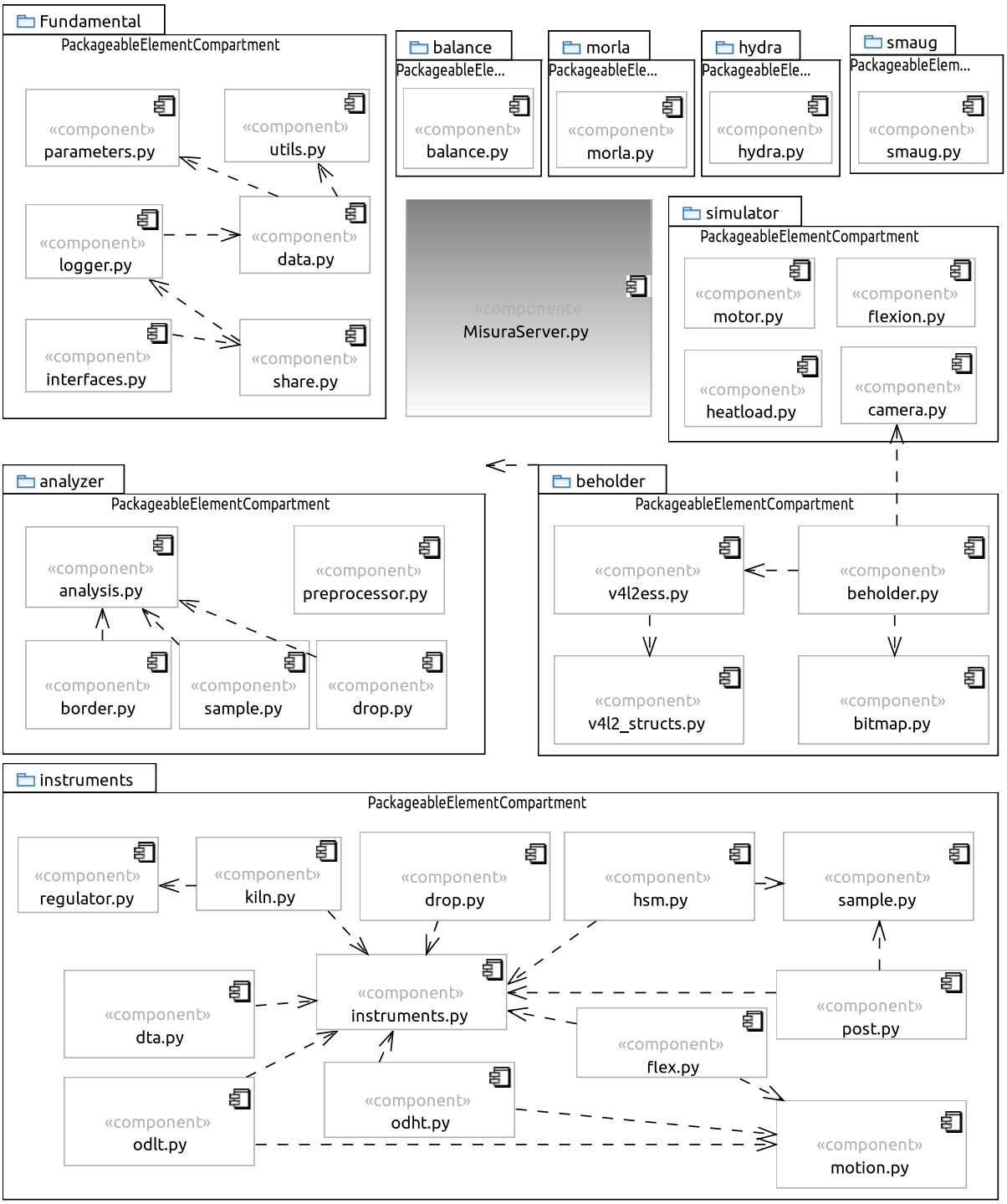
Algorithm 3.1 The simple multiprocessing API

```
from multiprocessing import Process

def f(name):
    print 'hello', name

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

Figure 3.1: Misura Server UML Component Diagram



between processes. The simplest ones are `Queue` and `Pipe`, which really are "communications channels": objects must be explicitly put into them in order for the involved processes to receive them.

Memory sharing constructs are much more useful: for example, `Value` and `Array` represent objects whose changes are immediately available to all involved processes.

An higher level of sharing is obtained by starting a dedicated server process, called `Manager`, which holds python objects (`dict,list,Value,Array`) and takes care of synchronizing them amongst processes which access for read/write operation.

As Misura needs to share complex data types, a custom Manager was implemented in share.py, the SharingManager, able to serve the following custom objects:

- `CircularBuffer`, an extended implementation of a CircularBuffer
- `Conf` objects, for configuration data management
- `SharedLogger`, for unified logging across processes

These classes are explained in following subsections.

A parallel `SharingManager` process is run to allow future processes to share data as the Server is started, and it keeps running for the entire execution of the Server.

The main process is a Twisted `reactor` (page on page 58), which constitutes the core event loop of the Server. It provides network communication, threading and event dispatching.

The `reactor` is used to run, in separate threads, some general utility scheduled jobs (as status checks, database re-indexing, log file compression), and also some short acquisition-related jobs, like waiting for a motor movement to complete.

The main task of the `reactor` is to publish on the network all methods provided by a `MisuraServer` object and make them reachable via XMLRPC calls, so that Client interaction is possible through a secured SSL connection.

The XMLRPC protocol functionality is inherited from another Twisted class, `twisted.web.xmlrpc.XMLRPC`, which is parent of many of classes in Misura. The powerful Twisted module allows to build an xmlrpc-aware class by simply inheriting it and prefixing any public function name with the special string "`xmlrpc_`" (Algorithm 3.2).

3.2 The Data Classes

Fundamental `data` and configuration-related classes are contained in the data module, displayed in UML Class Diagram Figure 3.2.

3.2.1 CircularBuffer fixed size in-memory storage

The circular buffer is a common concept in computer science. It refers to a memory structure where data can be appended and retrieved up to a maximum capacity. After the limit has been reached, new datum is written over the memory space

Algorithm 3.2 Creating an XMLRPC-aware class

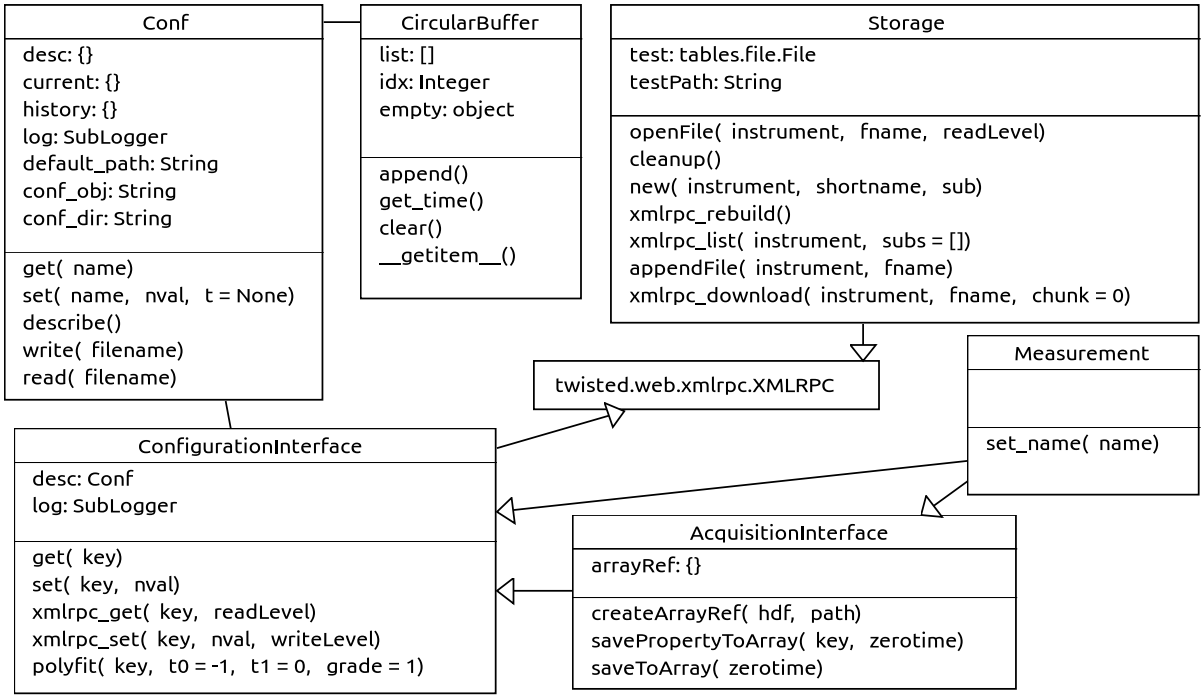
```
from twisted.web.xmlrpc import XMLRPC
class MyClass(XMLRPC):
        """Custom class inheriting XMLRPC"""
        def __init__(self):
                XMLRPC.__init__(self)
        def xmlrpc_echo(self,msg):
                """Function published through xmlrpc"""
                return msg
        def privatefunc(self):
                """Function not starting with xmlrpc_ are not exposed"""
                pass
```

Client side, the xmlrpc_echo function can be called by connecting to the xmlrpc server (http://server/RPC) publishing a MyClass object:

```
> from xmlrpclib import ServerProxy
> srv=ServerProxy('http://server/RPC')
> srv.echo("hello")
hello
>
```

The Python standard library xmlrpclib is used, as XMLRPC protocol is standard and can be accessed from any library or language implementing it.

Figure 3.2: `data` UML Class Diagram



previously occupied by the oldest datum in the structure. The `CircularBuffer` behavior resembles the Python `list` datatype by supporting the `append()` and the `__getitem__()` methods.

If the appended datum is itself a `list` or `array` type, and the first element is a `float` number, then the `CircularBuffer` interprets the first number of the list as a sequence number which increases with each `append()` call (for example, a time). In this situation, `h_get` method becomes effective (*history get*), making it possible to search for the value corresponding to a given time.

`CircualrBuffer` is the in-memory structure of all acquired data, of log messages, and of Misura client-server event signaling.

A `CircularBuffer` instance allows any new Client connecting to the server to retrieve the last messages, and keeps them updated by replying about new messages emitted from the last query time-stamp.

During acquisition, `CircularBuffer` instances are queried in order to retrieve new values to be written on disk, and that's how most of the Misura output is built.

3.2.2 The Conf high-performance key-value database and the History mechanism

The `Conf` class is responsible of the management of all configuration options and acquisition values: **any** configurable parameter or output value is stored inside a `Conf` object. The `SharingManager` process instantiates all `Conf` objects, allowing current values to be shared amongst all other processes.

It can be considered the very minimalistic runtime database of Misura 4, optimized for speed and concurrent access and capable of containing only simple key-value data, referred as *options*.

Each key is represented by two standard `dict` dictionaries: `current` and `desc`. The first one only stores the current values: `current={key1:val1,...}`. The second one stores all key metadata inside sub-dictionaries (Algorithm 3.3).

The option value is read/written from/to the `current` dictionary, while the `Conf.desc` is used for validation and for Client graphical representation.

Figure 3.3 exposes the History mechanism managed by the `Conf` class, which keeps record of changes occurred to options.

Every time an option `key` is changed via a `set` call, its corresponding `attr` list is retrieved from the `desc` dictionary. The `attr` list holds special property flags, represented by strings as "ReadOnly", "Runtime", "History", etc.

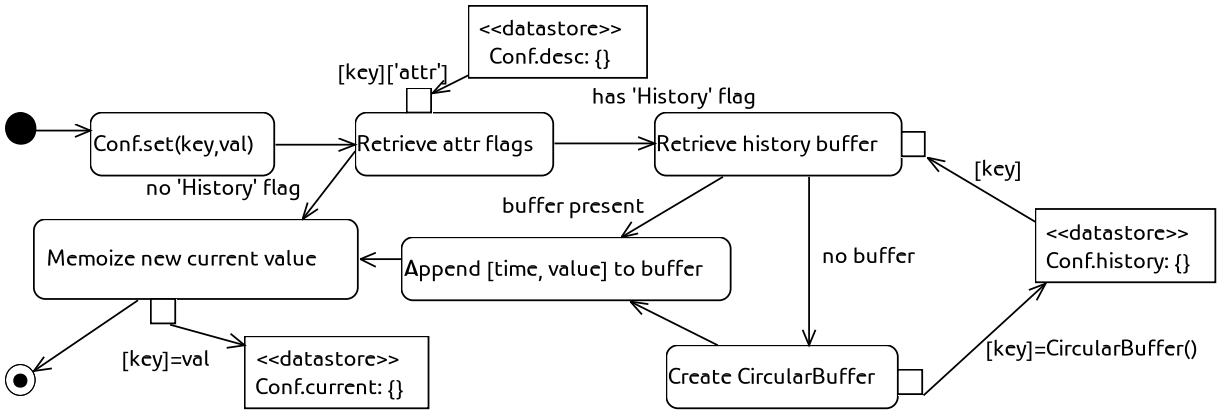
The "History" `attr` flag asks `Conf` to keep a chronology of all changes in the option value, in a corresponding `CircularBuffer` stored in the `history` dictionary: `history={key:CircularBuffer()}`. Only options with a "History" flag own a `CircularBuffer` referenced in `Conf.history`.

The `Conf` class implements a simple data persistence model, which consists in the ability to read and write CSV (Comma Separed Value) text files containing all stored data (`read` and `write` methods). Each `Conf` object refers to a read-only template preset, `default_path`, from which factory default values and metadata are loaded,

Algorithm 3.3 Option metadata

```
desc={key1:{"handle":key1,
                              'name':description...,
                              'factory_default':v,
                              'type':t,
                              'minimum':min,
                              'maximum':max,
                              'step':st,
                              'attr':['attr1','attr2',...],
                              'options':['opt1','opt2','opt3'],
                              'values':[v1,v2,v3],
                              'readLevel':r,
                              'writeLevel':w,
                              'priority':p,
                              'kid':k},
                key2:{"handle":key2,
                              ...}
                ...}
```

Figure 3.3: Conf.set UML Activity Diagram



and to a destination directory, conf_dir, where multiple versions of the data may be saved with different names.

This allows any Conf object to save different configuration profiles and subsequently load them. The active filename is referred in the conf_obj property.

For example, a microscope camera may save two profiles: one for low magnification and 6x6mm samples working condition, another for high magnification and 3x2 samples functioning.

3.2.3 ConfigurationInterface: advancing Conf to network, authentication, and more.

ConfigurationInterface objects constitute the network gateway for data stored in Conf instances. They inherit the machinery to speak the XMLRPC protocol from twisted.web.xmlrpc.XMLRPC. A consistent subset of the standard Python dictionary interface is published for Client access, like __getitem__, has_key, keys, __len__, etc. As a result, any ConfigurationInterface-derived object defined on the Server may be accessed from the Client as if it was a standard Python dict.

Figure 3.4 summarizes the activity followed by a set(key,val) call initiated by the Client. After having validated login credentials, the Server retrieves the current user's writeLevel authorization level from the user database. This writeLevel is passed as argument to the xmlrpc_set function, where it is compared with the authorization level needed to change the requested key. The key's writeLevel is stored in configuration metadata, in the Conf attribute ConfigurationInterface.desc. User's and key levels are compared in order to forbid access to unauthorized users.

After the authorization is given, the control passes to the set(key,val) function. Here the method list pertaining to the current ConfigurationInterface-derived object is consulted. If a special method named set_<key> is found in the object, than it is executed with the new value as argument. For example, if key='name', then method set_name is called. Otherwise, the new value is directly applied to the option in Conf.

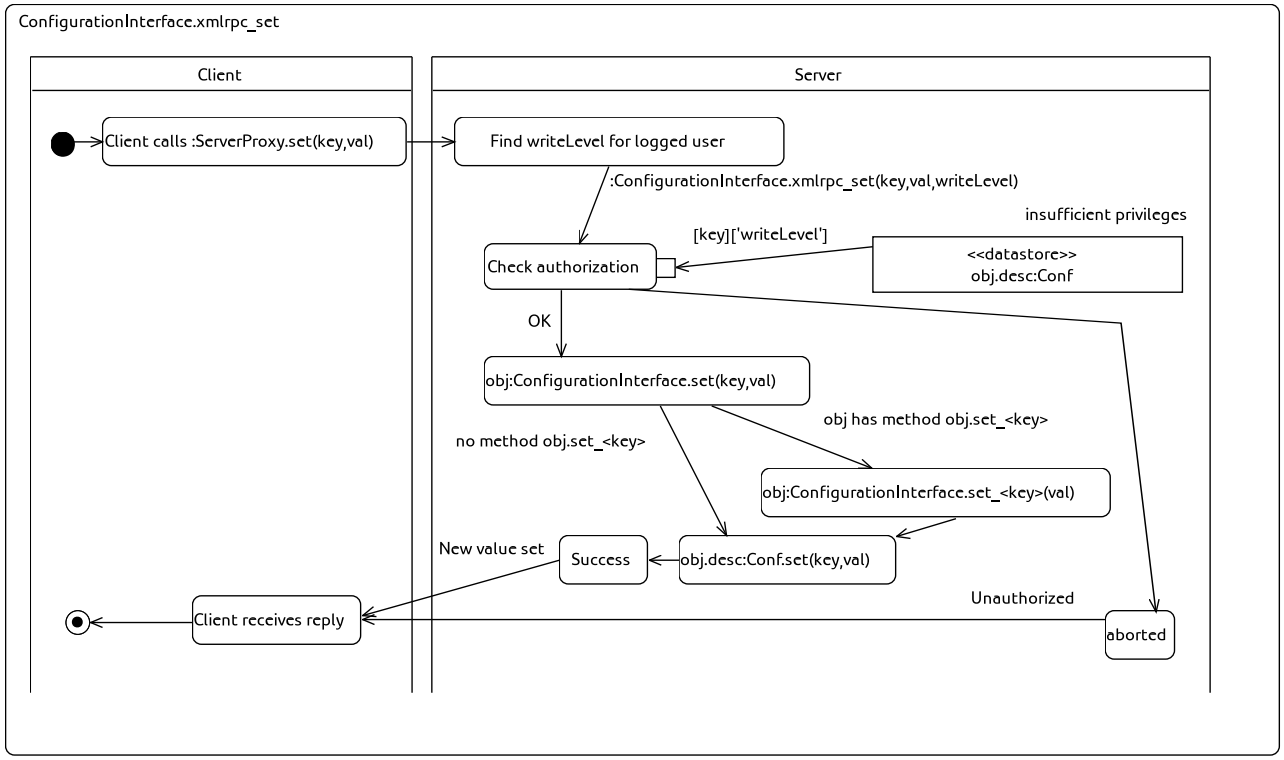
The graph is adequate for representing also a get(key) request, with the sole difference that readLevel is checked instead of writeLevel, and all method notations should be read as get : xmlrpc_get, get_<key>, get, etc.

This *stratification* allows ConfigurationInterface-derived classes to trigger custom function upon change/read request for any defined option. For example, when the special preset default option is changed to a new value, a ConfigurationInterface object will search for a CSV filename corresponding to the new preset and, if found, loads any configuration value found in it (using the Conf.load method).

3.2.4 Other data classes

The following Measurement and AcquisitionInterface classes are extremely simple, while the Storage class is still being developed so cannot be extensively described here.

Figure 3.4: `ConfigurationInterface.xmlrp` UML Activity Diagram



A short summary of their functionalities is presented in the following paragraphs.

Measurement for test filenames validation The `Measurement` class is an `AcquisitionInterface`-derive class, associated to a *test filename* where data will be/is being/was saved. Its implementation defines a `set_name(val)` method, which gets called when Clients execute `set("name",val)` (see Figure).

Inside this method, `val` is validated as filename: eg, special characters are removed.

AcquisitionInterface for data persistence The `AcquisitionInterface` class inherits `ConfigurationInterface` and adds three methods and one attribute. These methods are related to output operations towards test files on the hard disk.

The `createArrayRef(hdf,path)` method will create one output array for each option which has the "History" flag set and a numerical type (like Integer, Number, Float). The output array is created inside the `hdf` file, under the folder in `path` variables passed as arguments. A reference to the newly created arrays is kept in the `arrayRef=` dictionary attribute, using the option's handle as the key.

This method is called during the storage initialization, at the very beginning of a new test (see 3.5).

The `savePropertyToArray(key,zerotime=0)` method synchronizes the HDF array with new values contained in the `history[key]` corresponding `CircularBuffer`. The `zerotime` parameter will be subtracted from all time values, in order to consider the beginning of the test as 0.

This method is called during acquisition, in order to keep the test file updated with new data and avoid losing old data, which are overwritten by `CircularBuffer` internal management when the memory capacity has been reached.

The `saveToArray(zerotime=0)` is a shortcut method in order to call `savePropertyToArray` over all History-enabled options.

Storage for test files management, access and indexing The `Storage` class manages the Misura Server database. The data are stored in a file-system hierarchy. The main data folder contains one sub-folder for each instrument, and an HDF5 file containing the whole index.

Its principal tasks are:

- `new(instrument,shortname,sub)`: Create new test files in a coherent way, in `instrument/sub` folder, named after `shortname` or a derived name if already present.
- `cleanup()`: Remove old test files to limit database dimension.
- `xmlrpc_list(instrument,sub=[])`: returns the list of all tests produced by an instrument. It is used for Client browsing of database content.
- `xmlrpc_download(instrument,fname,chunk)`: Downloads a test file "chunk": test files are divided in small chunks so that downloads may be paused/resumed and a progress indicator can be implemented.
- `Storage` also serves results from the current test in progress (tabular data and images) to be plotted/visualized by the client. This works by opening the instrument's current test file, then reading data after a certain time asked by the Client, and returning it properly formatted.

`Storage` will not be further deepened here, as many parts of its implementation are still experimental.

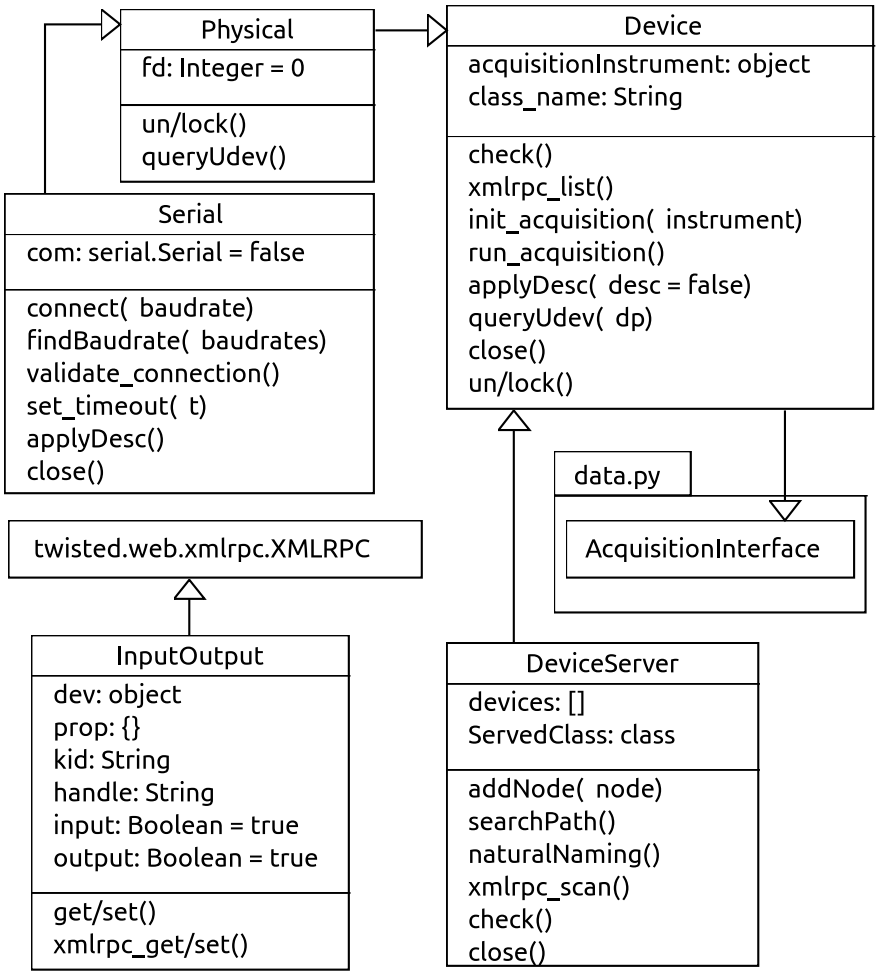
3.3 Devices and Device Servers

The `interface` module extends `data` classes in order to support acquisition, real devices representation and enumeration and configuration persistence organization.

Figure 3.5 presents the Class Diagram of the `interface` module: the main class `Device` inherits `data.AcquisitionInterface`. `Child classes Physical, Serial,` and `DeviceServer` further extend it. `InputOutput` only inherits `twisted.web.xmlrpc.XMLRPC` class for exposing simple get/set methods.

Many class methods are only defined as placeholders, which return formally correct values without doing anything real, except throwing an `UnImplemented` debug log message. Those placeholders are expected to be overridden in extensions classes.

Figure 3.5: `interfaces` module UML Class Diagram



3.3.1 Generalized Device concept

The `Device` class implements attributes and methods useful for acquisition, enumeration and data persistence file hierarchy organization.

Data persistence names

- `applyDesc(desc=False)`, which iterates all configuration options, triggering `set_<key>` methods for any option tagged with the 'Hardware' `attr` flag. Most common Hardware options effectively relate to external physical devices which need to be updated upon software configuration changes. But also pure software options may need to trigger `set_<key>` methods, in order to automatically update other parts of the object or of the entire application. An example are special `dev,devpath` options, which will be created upon configuration if nonexistent - overriding the default `ConfigurationInterface` behavior which is to throw an error.

- The `class_name` attribute is used to mark an instance's log messages and, under some circumstances, for determining the filename where to save the object configuration.

- `queryUdev(dp)` , will find the default configuration directory `conf_dir` and current configuration file `conf_obj` using the base configuration directory (set in `parameters` module), the `class_name`, the `'dp'` argument and, if set, the current `profile` option. Many child classes override this method, in part or completely.

Acquisition names

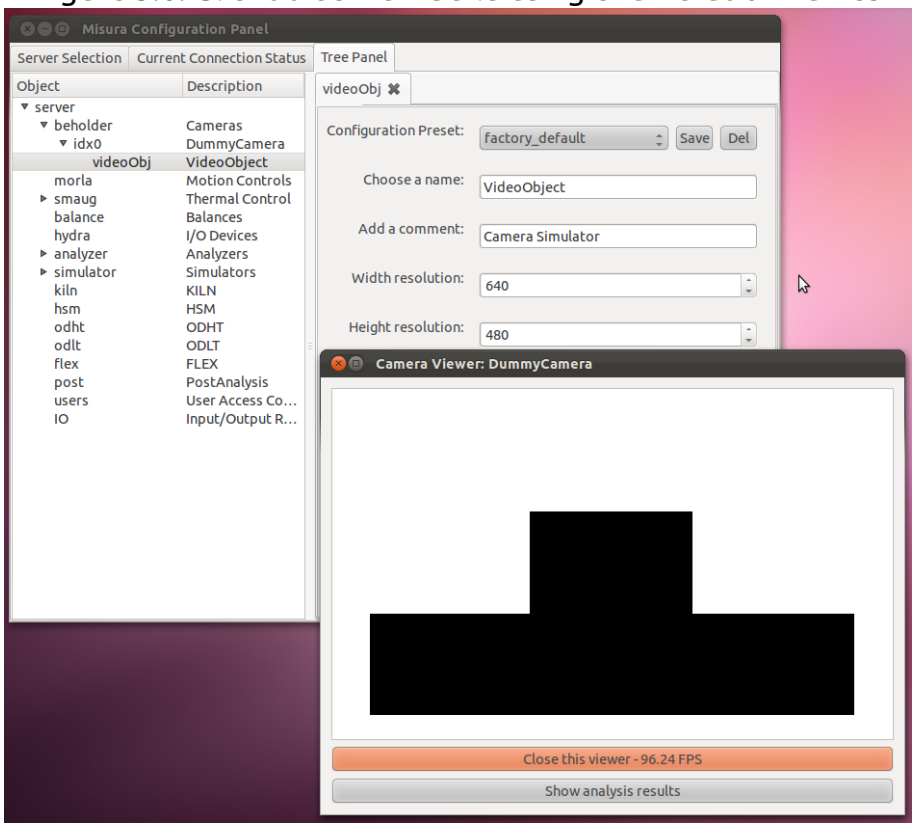
- `acquisitionInstrument` , attribute is a pointer to the Instrument currently using the object, and is set right at the beginning of an acquisition process.

- `init_acquisition` placeholder: this method should prepare the object for acquisition. Each subclass will implement this method depending on its specificity. The placeholder method simply sets a `title` attribute used to describe folders owned by the object in the test file.

- `run_acquisition` is another placeholder method. Subclasses will override this method to contain the acquisition loop of the device, to be run on a separate process during acquisition. For example, a subclass representing a balance will insert a loop reading the current weight and setting it in an History-enabled option.

- `lock()/unlock()` methods are used to prevent concurrent access to options or sensible device parts.

Figure 3.6: Client tree view build using enumeration names



Enumeration names A device can own child objects by implementing the `xmlrpc_list` method, listing all child objects for Client display. For example, a `DummyCamera` object, used to emulate a real `Camera` behavior, owns a `VideoObject` which determines the image which is drawn during simulation (height, width and black shape profile). On the other hand, each child `Device` object has an `idx` option which holds the sequence number in the parent listing (in the `VideoObject` example, `idx=0` as it's the first and only child device).

The Client uses this information to reach devices and to display them in a hierarchical tree view, in Figure 3.6.

3.3.2 Real peripherals: Physical and Serial

Those two classes deal with real peripherals connected to the computer. `Physical` inherits `Device` and extends it by adding:

- Device file locking feature at the operative system level (overriding `lock(),unlock()`). In this way the device file can be locked for the entire operative system during sensible procedures or for the entire duration of the Server instance.

- `queryUdev()` is extended in order to determine the `usbVendor`, `usbProduct`, `serialNumber` of the peripheral and the entire USB connection path. The serial number or an encoded connection path string are used as unique identifier for

the device (also referred as devpath). The serial number enables the system to keep a device identification even if its USB port is changed.

After having found an unique identifier for the device, queryUdev() will call parent class Device.queryUdev(dp) method passing devpath as argument, and properly setting conf_dir and conf_obj for configuration persistence.

Suppose for example a balance with serial number "12345" is found. The queryUdev method will find this number, then set conf_dir to something like <basedir>/balance/serial_1234 and conf_obj to conf/default.csv. All subsequently saved configurations for this unique balance will be saved and loaded from that directory.

Serial further extends Physical by adding the serial communication machinery. The validate_connection placeholder method is overridden by derived classes, including some custom communications. Connection validation works by sending requests in device-specific protocols, and evaluating the replies consistency.

Validation allows to discard faulty devices and, by duck testing[Mazey, 1946], to automatically find which kind of device is connected to a port: *if the device replies as a balance would, than it is a balance*.

3.3.3 Device organization: DeviceServer

The DeviceServer class, derived from Device, is conceived to find, group and index devices pertaining to the same family.

The device family is a reference to the class of objects served by a DeviceServer, stored in ServedClass attribute. For example, a DeviceServer which serves camera objects will have a reference to the Camera class stored in its ServedClass attribute.

Each DeviceServer subclass will implement a get_rescan method which searches supported peripherals. The addNode(node) method is called for each compatible device file node found (eg, /dev/video0, /dev/video1, etc), which in its turn tries to instantiate a ServedClass object passing the node in the class initializer. If the object instantiation is successful, the DeviceServer registers it as child object.

DeviceServer also offers a searchPath method for finding the object representing a certain devpath. This method is particularly useful for instruments classes, which assign acquisition roles to devices by storing their unique devpath identifier. For example, Hsm microscope instrument class will store in the camera option a reference to the unique identifier of the specific device used to get images (eg, serial_54321). It will then call searchPath from the DeviceServer serving all connected cameras in order to retrieve the object, where obj['devpath']=serial_54321.

There is one DeviceServer for each kind of peripheral which can be currently used by Misura. They are defined inside separate packages (Figure 3.1), in sub-directories with all necessary Device/Physical/Serial classes and service modules.

Packages are named after mythology or fantasy literature creatures.

- Beholder class in beholder/beholder.py: imaging-related devices. It currently deals with all Video4Linux 2 compatible devices.

- `Morla` class in morla/morla.py: stepper motors control boards.
- `Smaug` class in smaug/smaug.py: temperature/heat/power-related device.
- `Hydra` class in hydra/hydra.py: generic I/O devices.
- `Balance` class in balance/balance.py: weighting systems.

More `DeviceServer` classes are defined for non-hardware related objects:

- `Analyzer` in analyzer/__init__.py: image analyzers
- `Simulator` in simulator/__init__.py: software simulation of hardware-related classes, useful for testing or demonstration purposes.

3.3.3.1 Input/Output abstraction

An `InputOutput` object (I/O) represents a single option which can be accessed ignoring the object who owns it and the corresponding option name. The `InputOutput` takes care of routing `get()/set()` calls in the correct way.

Consider, for example, a multichannel acquisition device, like a multiple thermocouple reader which exposes 3 temperature readouts. Each channel will have its corresponding option in the `Serial` device representing the peripheral: `T1, T2, T3`, etc. Connected to the same instrument there is also a thermoregulator device, which exposes its own `temp` option and also a `power` option for setting the output power percentage to the power unit.

Instruments can configure how (and if) to use these I/O options, by assigning them an acquisition Role (see 3.4). For example, `temp and T1` may be connected to thermocouples near the samples, while `T3` may be a security sensor near the heating elements and `T2` is left unused, following any disposition of real sensors in the equipment.

Instruments objects will then use those I/O get()/set() methods without caring from which device they come and which option name they have on that device.

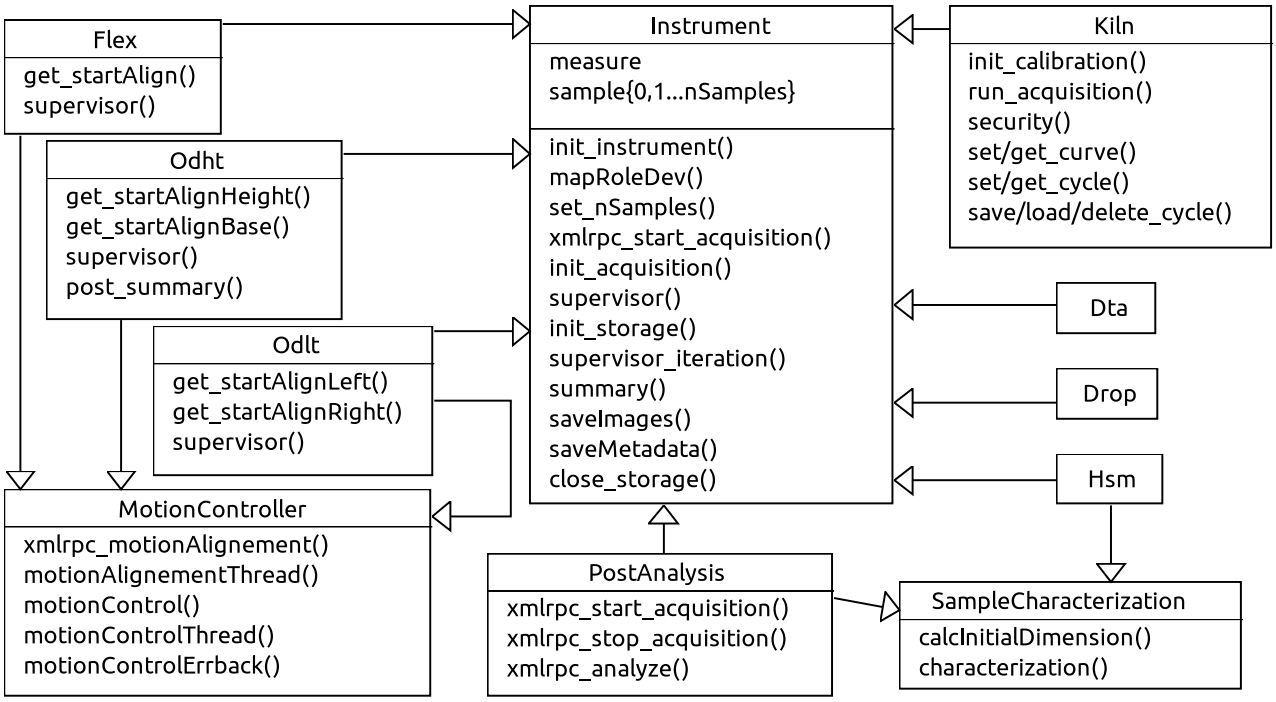
3.4 Instruments

The `instruments` module organization is described by the Class Diagram in Figure 3.7.

A base Instrument class is defined inside the instruments.py module. It collects all aspects related to:

- Managing devices involved in acquisition, according to the Device/Role mechanism.
- Collecting and organizing test and sample metadata.
- Initializing a proper file storage hierarchy in the HDF5 output file related to a single test, and closing it at the end of the test.
- Conducting the Acquisition Orchestration, and monitoring its evolution until termination conditions are met or the client asks for acquisition stopping.

Figure 3.7: `instruments` module UML Class Diagram



3.4.1 The Device/Role mechanism

To allow maximum flexibility in instrument configuration, realization and research activities, devices are rarely connected in a predetermined and stable disposition. Thus, a method for flexibly mapping devices to tasks to be accomplished during an instrument acquisition was conceived.

Each instrument will enlist, amongst its configuration options, some special Role options, referred to a task which needs a device to be completed. These Roles will refer to an unique device path, which will be searched using `DeviceServer.searchPath` function in all expected device servers. During the instrument initialization, triggered by the client or before an acquisition starts, all those Roles are mapped to existing devices.

A simple example will better clarify the concept. A `Flex` instrument defines a camera option of type `RoleBeholder`, and a motor option of type RoleMorla. The user should set `camera` option to the `devpath` of the vision peripheral effectively seeing the fleximeter sample, and the `motor` option to the `devpath` of the motor moving that camera.

Role options also require a device configuration `preset` name, which will be loaded upon device initialization. Suppose the same camera is used by both the `Flex` and the `Hsm` instrument. The configuration preset associated with the `Hsm RoleBeholder` option may have a lower magnification value set, that would be applied automatically to the device when the instrument initializes.

There is one role option type for each real-peripheral DeviceServer: `RoleBeholder`, `RoleSmaug`, `RoleMorla`, `RoleHydra`, `RoleBalance`.

Beside Role options, there are IO options, which maps not only to a device and

a preset name, but also to a very specific option of that device (see 3.3.3.1). Very frequently an instrument only needs a single value to be read/written, for example a temperature, a digital on/off signal, etc. Input/Output options are identified by an option type corresponding to `RoleDeviceServerIO`, for example `RoleSmaugIO`, `RoleHydraIO`, `RoleBalanceIO`, etc.

The `mapRoleDev` method takes care of translating Role, `devpath` and option key information into references to `Device` objects or new `InputOutput` objects, which will be saved as new attributes of the `Instrument` instance.

3.4.2 Metadata sub-objects

Every instrument has a `measurement` attribute, instance of the class `Measurement`, which holds all test metadata: title, description, start time, elapsed time, acquisition options, stopping conditions.

The special option `nSamples` holds the number of concurrent samples analyzed, and triggers the instrument to automatically add a new sub-object for each sample: `sample0`, `sample1`, `sample2`, etc. This currently only applies to microscopy instruments, but is generalized anyway in the `Instrument` class.

Those sample object are plain `AcquisitionInterface` instances which store metadata about the analyzed sample, like name, material, and all current analysis results. For example, an Hsm sample object would have options like `h` (height), `w` (width) etc, which are continuously updated during the acquisition process, and saved to the test file via `AcquisitionInterface.saveToArray` calls.

3.4.3 Test file hierarchy

The output of a test run is an HDF5 file, which is organized in a similar way as a filesystem: there is a root folder, child folders which can have further children, and dataset objects, which cannot have children. Misura uses three types of datasets:

- Enlargeable arrays, used to store homogeneous multidimensional data which can be expanded by appending new points.
- Tables, used to store non-homogeneous data in a table structure.
- Filenodes, for storing generic binary files, like images or raw data retrieved by special peripherals.

The test file hierarchy is determined and initialized by an Instrument object just before entering the acquisition loop, which will then append acquired data to predetermined positions in that structure.

The base hierarchy is automatically determined following devices having a Role in the acquisition (generally referred as *acquisition objects*): defined samples, the `measure` object, and the instrument object itself. For every 'History' option defined in these acquisition objects, a corresponding array is created in a proper location in the hierarchy. Option arrays contain [time,value] points, storing the value of the option

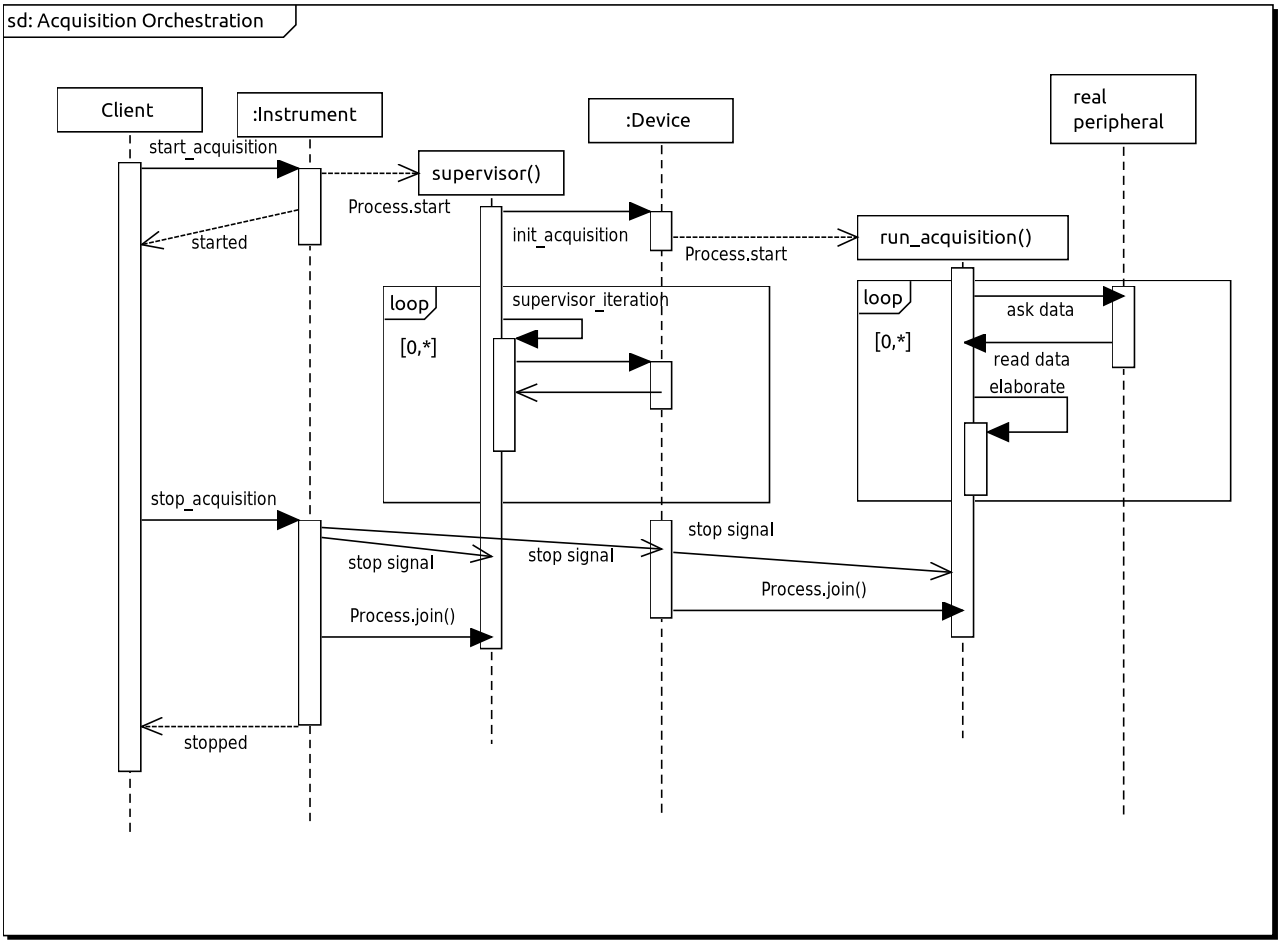
after every change occurred during acquisition, and the time at which that change was made (see 3.3).

The resulting structure is organized as follows:

- `cycle`: the thermal cycle is stored as a table with columns for time [float], temperature [float], checkpoint flag [boolean].
- `log`: all log messages generated during the acquisition are stored in this table, with columns time [float], priority [integer], owner [string], message [string]
- `conf`: folder containing one table for each acquisition device, sample, and for the measure object (generally referred as acquisition objects).
  - {*N} tables storing the complete configuration option listing of each object involved in the test, with one column for each option metadata field (defined in 3.3).
- `dev`: folder containing one sub-folder for each acquisition device.
  - {*N} acquisition object sub folder
    - {*M} one enlargeable array for each object option with 'History' attribute set, where the chronology of all changes occured during the acquisition is saved.
- `samples`: folder containing one sub-folder for each defined sample.
  - {*`nSamples`} sample object subfolder
    - {*M} one enlargeable array for each sample option with 'History' attribute set
- `dat`: folder containing one sub-folder for each device requiring raw data storage.
  - {*N} special device object subfolder
    - {*i} binary data filenodes saved during acquisition, usually with a progressive number in their names for identifying their sequence.
- `summary`: a wide table grouping together *all* acquired data, interpolated around 2 seconds intervals. Column names refer to all History options set in *all* acquisition objects. It is created for client consultation during acquisition, to avoid huge amounts of raw data to be transmitted.

At the end of a test, all tables contained in the `conf` folder are updated to the current configuration of objects at the end of the acquisition.

Figure 3.8: Acquisition UML Sequence Diagram



3.5 The Acquisition Orchestration.

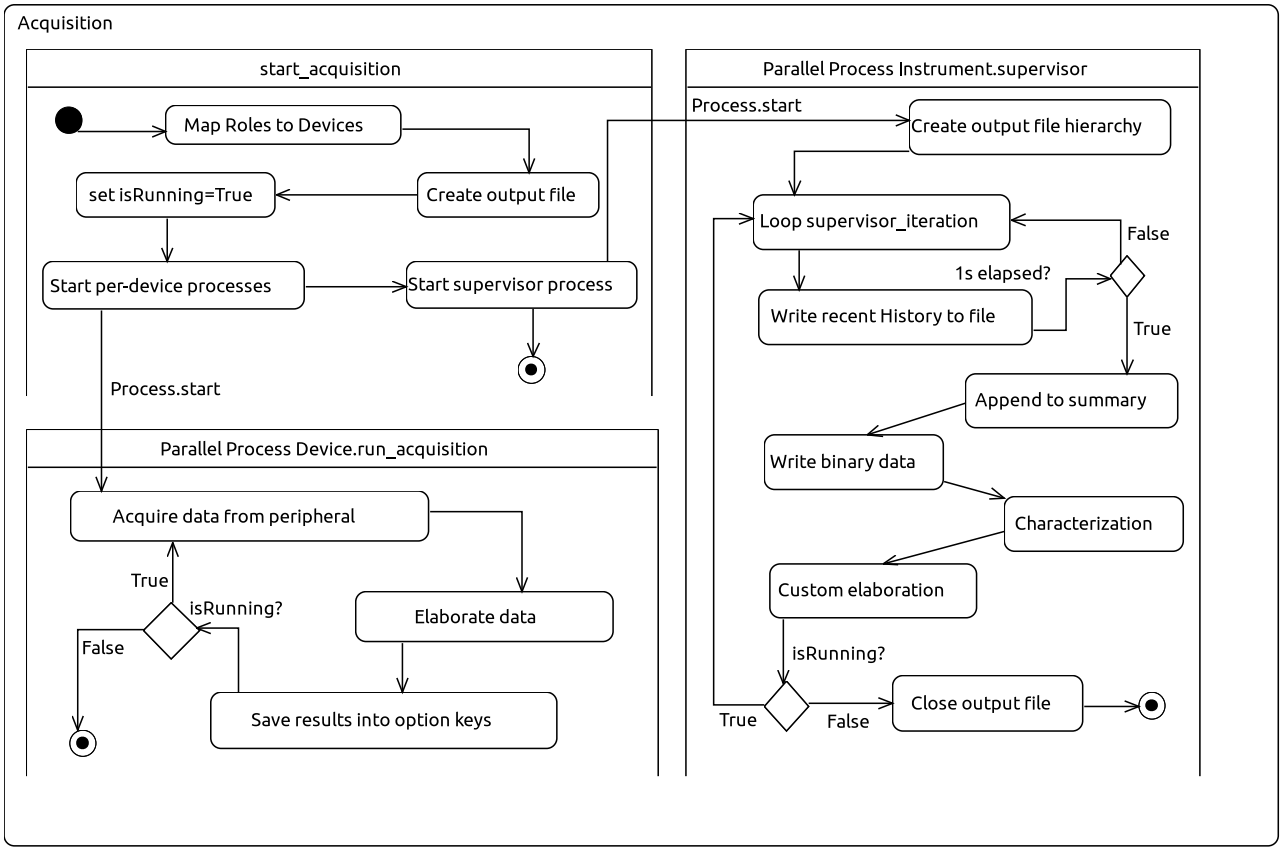
The parallel nature of Misura 4 is completely enforced during acquisition. In section 3.1 were already introduced the two main, permanent processes which constitute Misura Server: the Twisted `reactor`, handling communications with Clients, and the `SharingManager`, managing the shared memory space. When the Client asks for a new test to start, calling the `start_acquisition` method of an Instrument object, a cascade of events is ignited on the `reactor` main process, modeled in the Sequence Diagram in Figure 3.8.

The Client asks for a new test to start, calling the `xmlrpc_start_acquisition` method of the instrument which will be performing the measurement. The instrument immediately starts a new parallel process, where its own `supervisor()` method is executed. Each device having a role will start a new parallel process executing its `run_acquisition()` method.

Thus, during the acquisition there will be these parallel processes:

- The main Twisted `reactor`, collecting requests to start/stop the process and sending status and results to the client.
- The `SharingManager` for synchronizing data amongst other processes.

Figure 3.9: Acquisition Initialization UML Activity Diagram



- The `Instrument.supervisor()` which continuously collects and writes data to the output file.

- One `Device.run_acquisition()` process for each acquisition device object having a Role in the instrument, which usually deals with real peripherals.

The Activity Diagram in Figure 3.9 further details the actions performed during the initialization and the acquisition itself.

The first action in `start_acquisition` method is to update Role-to-Device mapping, by calling the `Instrument.mapRoleDev`. The device mapping will remain fixed for the entire duration of the acquisition process, but will be modifiable again after the acquisition finishes.

The output file is then created: `Storage.new` is invoked and, based on `measure['name']` option value, a new, unique filename is determined.

The `'isRunning'` option defined for the `MainServer` class store the current analysis status: if there is an `Instrument` currently in the acquisition process, it is `True`, otherwise, `False`. Acquisition-related processes will continuously read this value in order to decide if to continue looping or to terminate themselves. So it is set to `True`, and then per-devices `Device.run_acquisition()` and `Instrument.supervisor()` methods are executed in new parallel processes.

Processes are modeled as *regions* in the State Diagram (Figure 3.9): the bottom region holds a generic run_acquisition() process, the right region holds the supervisor()

process.

3.5.1 Per-device processes

The run_acquisition() method of devices usually involves some initialization code, and then a `while server['isRunning']` loop which executes device-specific code for dealing with real peripherals. For example, a Camera object will open the device, then continuously get a new frame and analyze it. This iterates until the isRunning option remains True, or a fatal event stops the device. Although a device process may prematurely exit, this will not usually affect the rest of the analysis. The supervisor process will catch that event, and judge if and when to abort the whole measurement.

For example, the special Kiln object (which can both act as an Instrument or as an acquisition device for any other Instrument), will exit its run_acquisition() process when the thermal cycle has ended.

In the same way, devices may also forcefully block all other acquisition processes just by setting the `isRunning` flag to `False`, usually after a fatal event.

The above mentioned Kiln object will throw this fatal exception if the temperature condition inside the Kiln chamber is anomalous or dangerous.

3.5.2 The Supervisor process

The already created empty HDF5 output file is populated, recreating the hierarchy explained in section 3.4.3. Then the process enters the `while server['isRunning']: ...` loop which continuously executes the `supervisor_iteration` method.

The recent chronology of each option having the 'History' attr, amongst each acquisition related object is queried, and saved to the output file.

A new point is interpolated around 1 second before, from all History options, and appended to the summary table. If present, any binary datum is also saved (currently only images and shape profiles).

The characterization method will calculate cross-option metadata and identify analysis phases. A dilatometry instrument calculates the sample expansion by adding border movements acquired from multiple cameras (stored on `'d'` options owned by each device). A microscopy instrument will guess current characteristic shape based on whole analysis behavior with respect to time and temperature.

During the characterization the instrument will also check for termination condition, and automatically sets `isRunning` to `False` if one condition is met.

Children instrument, which inherits the general `Instrument` class, may override the supervisor method, as modeled in Diagram 3.7. For example, all `MotionController` children instruments override the `supervisor` method. After having performed the standard `supervisor_iteration`, those instruments will call the `MotionController.motionControl()` method in order to automatically move cameras according to sample movements.

Stopping the acquisition The Client may ask to stop the acquisition in any moment, by setting the `isRunning` option to `False` or by calling `Instru-`

`ment.xmlrpc_stop_acquisition` on the active instrument. An optional `False` parameter may be passed to the latter function for discard the measurement, and immediately remove the test file from the storage. Otherwise, the file is always kept, and may be subsequently deleted from the storage.

In above paragraph two more events which may stop the acquisition are listed:

- A device throws a fatal exception by forcefully setting isRunning to False.
- A device sub-process ends and the supervisor judges the measurement will no longer have any meaning without that device working.
- A termination condition is met an the test is automatically stopped.

Chapter 4

The Client Architecture

This Chapter describes how Client functionality is implemented in order to accomplish all required tasks with a high generalization and abstraction level.

Client tasks, introduced in section 2.6, are:

- Networking
- Configuration
- Acquisition
- Plotting
- Database
- Scripting
- Translation

The Client side of Misura 4 was designed to dynamically adapt to the characteristics of the Server side. The graphical user interface (GUI) will *build itself* based on how the Server describes itself or on the metadata documenting the measurement output files.

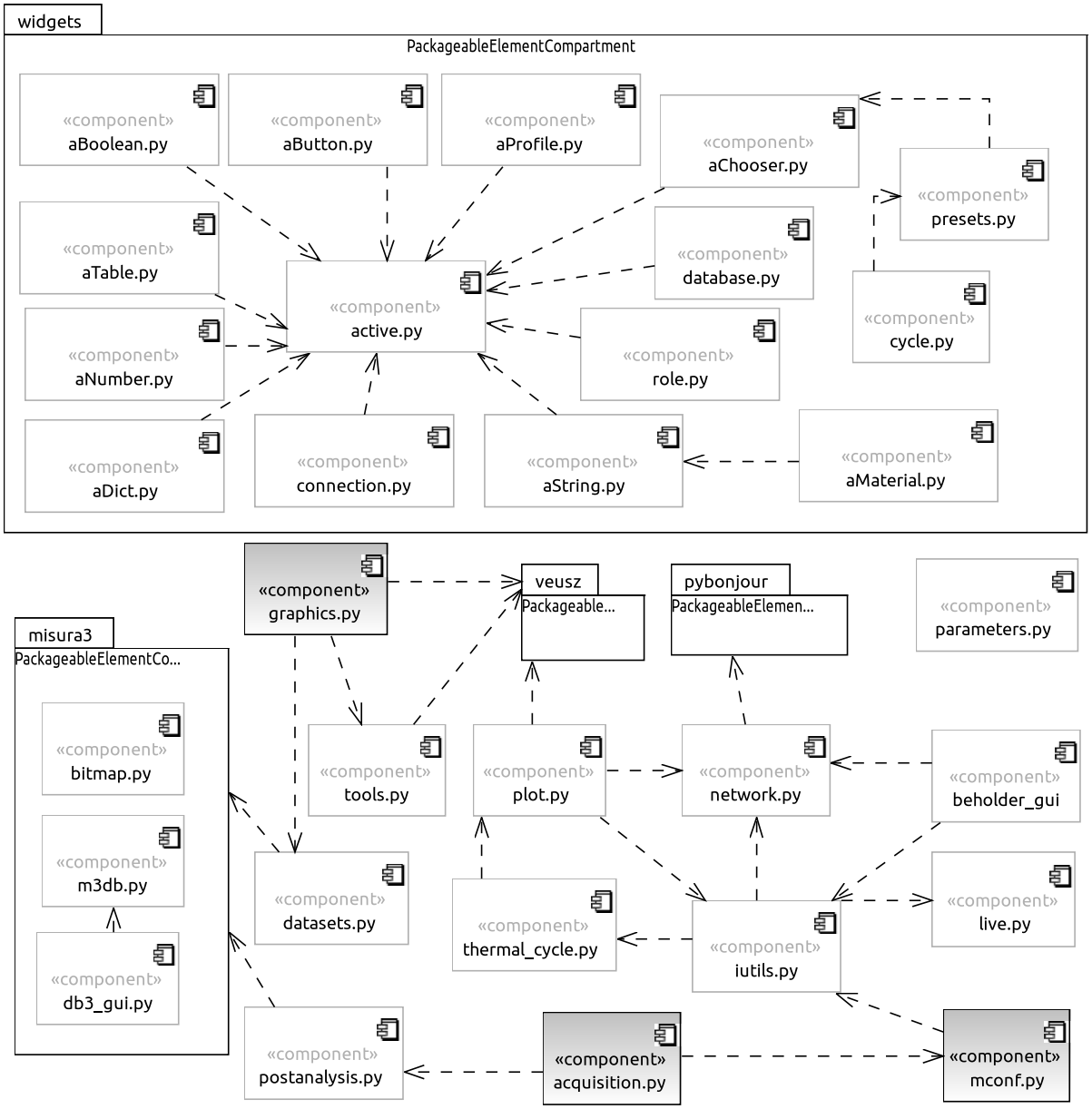
This method substantially reduces the development effort in synchronizing changes between Client and Server. As long as the Server API remains constant, no change in the Client code is needed. Vice-versa, since the Client continues relying on the same Server API, calling the same functions and expecting the same replies, no Server update is required.

As a result, while the Server is implemented in more than 30'000 lines of code, the Client counts just 8'000 lines.

Modules and packages constituting the Client are modeled in the Component Diagram 4.1. The diagram contains four packages, containing well-defined functionalities:

- The `widgets` package contains graphical representations of Server's remote *options*, presented in 3.2.2. Graphical elements (widgets) are also defined for connection and remote database access.

Figure 4.1: Misura Client UML Component Diagram

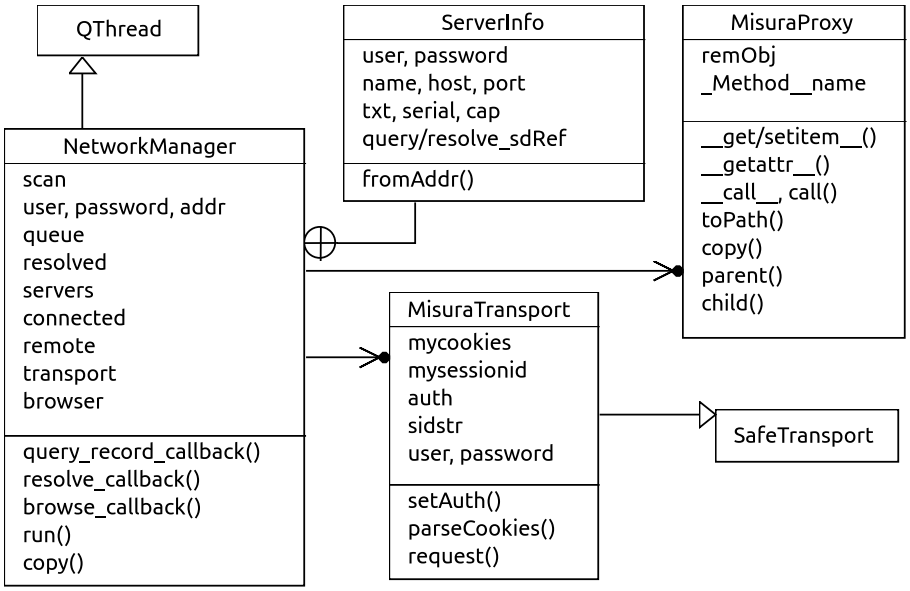


- `veusz` and `pybonjour` packages are third-party libraries. Veusz is used for plotting, pyBonjour for Server auto-discovery on the local network though the Multicast DNS standard.
- The `misura3` package supports read access to legacy Misura 3 database.

Execution entry points are evidenced by a gray background, and may be:

- `acquisition.py`, starting and monitoring actual measurements. This is the main entry point, started from proper Operative System shortcuts (Start menu entries or desktop links).
- `graphics.py`, giving access to measurement output files once the acquisition is finished. It can open local files, access to the Server storage or to legacy Misura 3 databases.

Figure 4.2: Network module Class Diagram



- `mconf.py` displays a general configuration panel for Server administration, useful for factory installation, maintenance and debugging purposes. This panel is also accessible from the acquisition module.

Client's mechanics is further explained in following subsections.

4.1 Networking Layer

The `network.py` module handles connections to a running Server instance.

A separate `QThread` is started as one of the entry points get executed, where a `NetworkManager` instance will scan the local network for available running Servers.

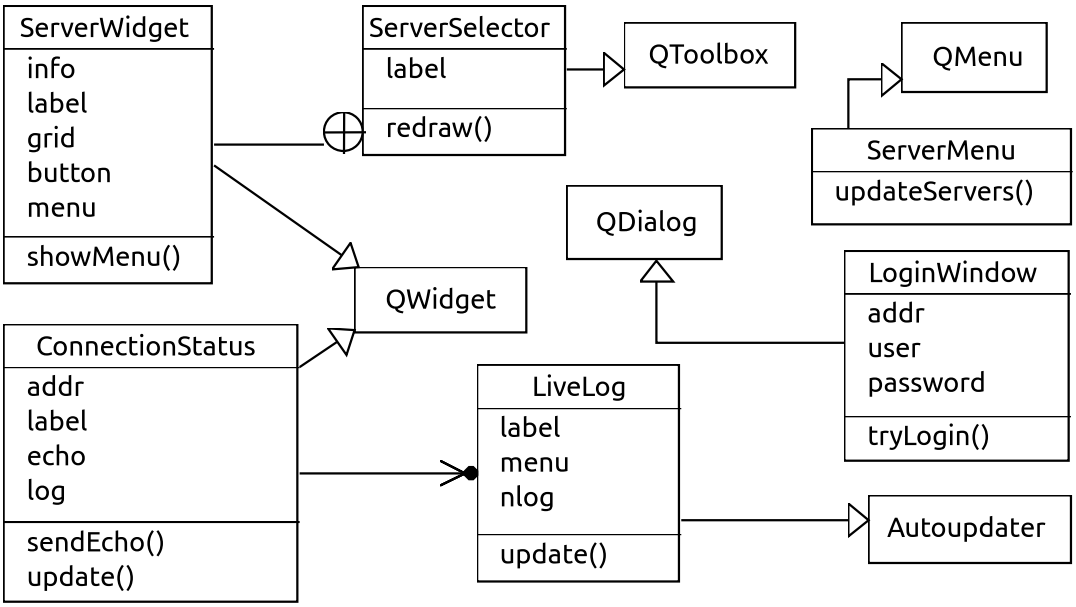
The auto-discovery of Server instances is demanded to the third-party library `pyBonjour`, implementing *Zeroconf*, the Zero Configuration Networking system. An instance of `pybonjour.DNSServiceBrowse` broadcasts query packets to the network, and decodes the replies. When a Misura 4 Server reply is recognized and decoded, a `ServerInfo` instance is built and stored in the manager.

`ServerInfo` contains the information needed to get a proper connection to a Server, together with some metadata. The metadata comprises:

- The serial number of the Server machine.
- Its capabilities and supported instruments (for combined models, HSM+Dilatometer, etc)
- If the Server was already known by the Client, username/password authentication keys are recovered from the registry.

When the user selects the Server and connects, the manager tries to establish an HTTPS secured connection through `MisuraTransport`.

Figure 4.3: Network Widgets Class Diagram



The Secure Layer Transport cryptography (SSL [Freier et al., 1996]) is then initiated by the `MisuraTransport`, which receives a new session identifier (SID) upon successful connection. The transport encodes current user name, password and unique SID strings into the `auth` MD5 hash [Rivest, 1992]. The `auth` ash is included in HTTP cookies, along with the user name. The Server verifies the identity of the Client with user and `auth` keys, without the need for the password to be ever transmitted over the network.

Once the SSL connection is active and the Client is authenticated to the server, a `MisuraProxy` object is created to allow transparent client access to remote objects and methods. Basically MisuraProxy routes all function calls to the remote server through a standard Python `xmlrpclib.ServerProxy`, stored as `remObj` attribute. Beyond basic `ServerProxy` functionalities, MisuraProxy adds some shortcuts which are extensively used to access specific Server methods:

- The Python dictionary interface is partly implemented through `__getitem__` and `__setitem__` methods, which are routed `remObj.get` and `remObj.set` calls. This interfaces enables MisuraProxy objects to be treated like dictionaries: `obj['key']=val`.

- `parent` and `child` methods, returning the parent object or a list of child objects. For examples, a remote `DeviceServer` object (like `Beholder`) has child objects (cameras) and one parent object (the main `MisuraServer` instance). These shortcuts allows easy reconstruction of remote hierarchy.

- `copy`, for cloning a proxy object, comprising connection status and information. Copying is necessary when the same `MisuraProxy` object must be accessed from concurrent threads.

The user actually interacts with these mechanics through the widgets modeled in Class Diagram 4.3. ServerWidget shows metadata information stored in a ServerInfo object.

`ServerMenu` and `ServerSelector` list all Servers found on the local network, by accessing the list of `ServerInfo` objects built by the `NetworkManager` thread. The first one inherits `QMenu` from the Qt Library (page 58), and looks like a menu with one entry per `ServerInfo` object.

`ServerSelector` is a toolbox containing `ServerWidget` instances, which display all `ServerInfo` metadata (except the password value).

When the user choose the Server he wants to connect to, the LoginWindow appears, asking for username and password.

`LiveLog` and `ConnectionStatus` are used for monitoring remote Server's status, by enumerating its recent log messages, allowing for echo requests to be sent and showing connection statistics.

4.1.1 Client-Server Synchronization

Once the Client has a connection and knows how to call remote methods, it could continuously query for current status *options* values in order to remain up-to-date with remote changes. This quickly becomes unproductive if the system must deals with thousands of remote objects to synchronize.

During Server discussion, a `kid` metadata value was associated to each option. The kid, key identifier, is uniquely associated to a specific option pertaining to an unique object instance.

For example, option "`name`" owned by a particular `Camera` object will have a different `kid` than the same option "`name`" owned by an `Instrument` instance.

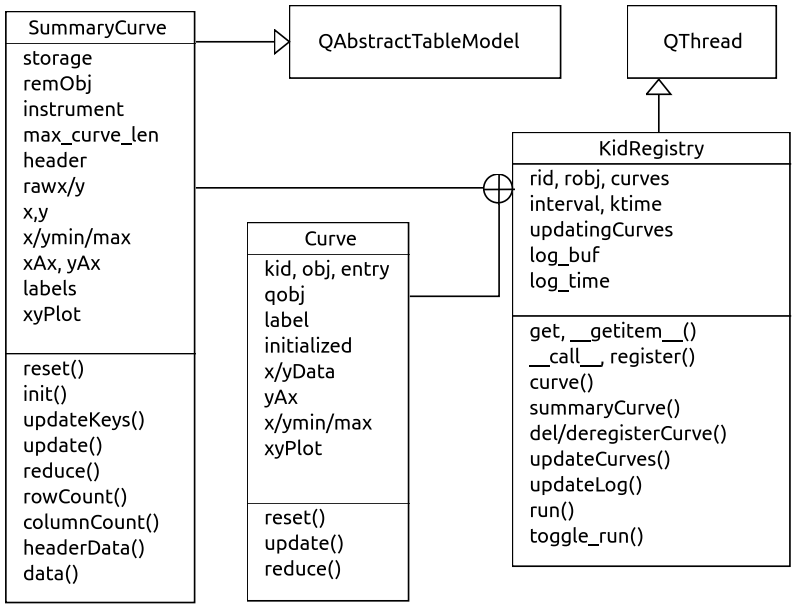
Unique Key Identifiers are the basis of the Client/Server synchronization. Each time an option changes on the Server its `kid` is appended to a registry of recently modified objects. The Client queries this registry and, if a displayed option is present, updates the proper widgets to reflect remote changes.

This continuous `kid` check is done in a separate thread, containing a `KidRegistry` instance. All graphical elements representing remote options register themselves into the KidRegistry. When the kid check cycle founds that a registered kid is present in the recently changed list, will throw an `update()` signal. Widgets will eventually catch that signal, and do the proper actions in order to update themselves.

Remote options may be recorded as `Curve` objects. This causes all recent changes to be tracked into a (time, value) sequence of points, for easy plotting.

During acquisition `Curve` objects would be too slow, as tens of points are produces each second and would translate in huge amount of data to be stored and displayed. Thus, a `SummaryCurve` object is defined, for direct access to the summary table of interpolated points, defined in 3.4.3.

Figure 4.4: Live module for data synchronization, Class Diagram



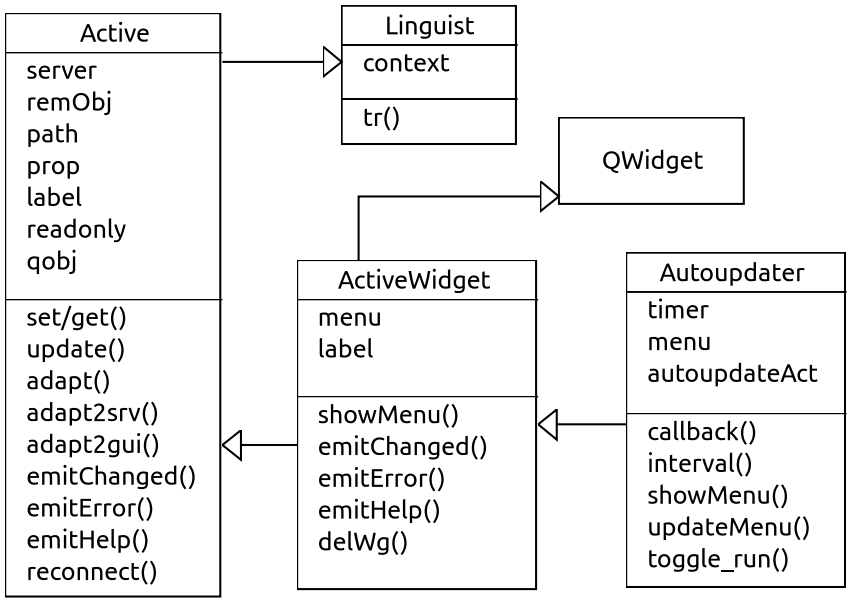
4.2 Active Widgets: The Building Blocks

Any object capable of interacting with a remote option is referred as *active*. The `Active` class implements basic functionalities shared amongst all *active* objects:

- Storing, inside a class attribute, references to:
  - `remObj`, the remote object owning the option (`MisuraProxy` instance)
  - `prop`, a dictionary containing option metadata
  - `label`, a descriptive label string
  - `qobj`, the `QObject` receiving update signals from the `KidRegistry` when `Active` is not subclassed.
- `set` and `get` the option current value
- react to remote changes. The `update` slot is connected to the update signal thrown by the `KidRegistry`.
- translate the values contained in the graphical representation to acceptable Server's value, and vice-versa (`adapt`, `adapt2srv`, `adapt2gui` methods).
- emit a standard set of signals when certain conditions are met (`emitChanged`, `emitError`, `emitHelp`). Those are just implemented as placeholders printing some text, since `Active` should not interact directly with the user.
- react to a connection reset (`reconnect`).

The `ActiveWidget` redefines some of `Active` functionalities or expand them, in order to create a graphical element for the remote option. The widget has a context

Figure 4.5: The Active Widget concept, Class Diagram



`menu`, a visual `label` and can delete itself from layouts (`delWg`). Signals are effectively emitted overridden in `emit-` methods.

An `Autoupdater` class furher extends `ActiveWidget` by adding the possibility to continuously check the current value - also if no change has been notified by `KidRegistry`.

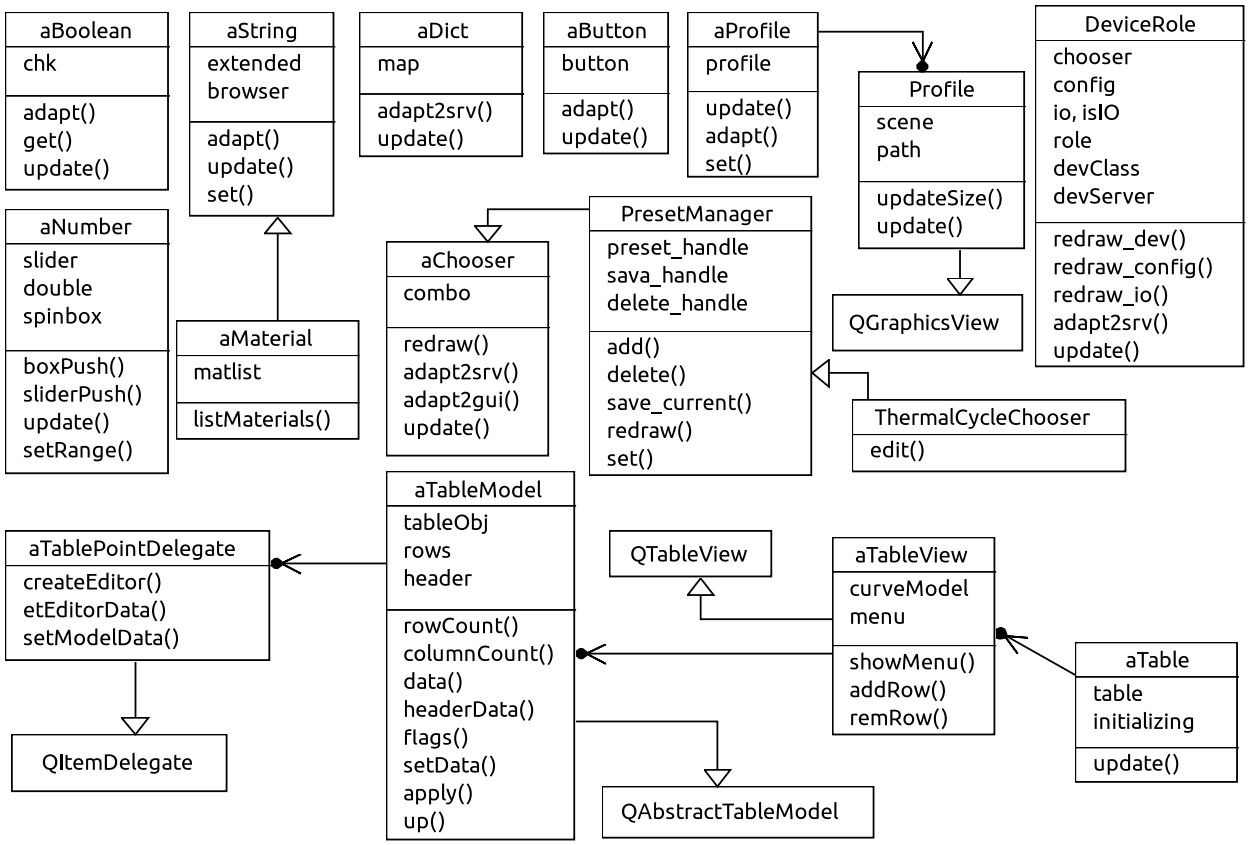
All strings visualized by the user are passed through the `tr` method of the `Linguist` class, parent to any other widget class. This method handles string translation and records untranslated strings to a special file, to ease the interface internationalization work.

4.2.1 Options

More complex graphical elements all inherit those basic functionalities, and are modeled in Class Diagram 4.6.

- `aBoolean` displays a simple True/False check box.
- `aNumber` displays an horizontal slider with an editable spin-box for reading and configuring a generic number (float, integer, etc). The slider is drawn only if both minimum are maximum are defined for the option.
- `aDict` represents a dictionary options with one label and one editable spin-box for each dictionary entry.
- `aButton` is the simplest widget, consisting in only a button holding the description of the option. When the button is pressed, the corresponding get() method is called for the option. `aButton` is used to get a simple client "action" representation. For example, the "`zero`" option of a balance will fire the `get("zero")`

Figure 4.6: Base Active Widgets Class Diagram



method. On the server the corresponding `get_zero()` call (Figure 3.4, page 82) will cause the balance to reset its current value.

- `aString` displays a single line editor or a text area editor, depending on the metadata of the option and the length of the string to be represented.

- `aMaterial` is similar to `aString`, and additionally let the user pick the value from a list of possible strings. This widget is used to list all materials analyzed in past tests, and allows the user to decide if picking an already defined material, or create a new one.

- `aProfile` plots the x,y curve contained in special "`Profile`"-type options. For example, the microscope sample profile can be plotted alone, without getting the full image frame.

- `aChooser` lets the user choose in a list of possible values, via a `QComboBox`.

- `PresetManager`, derived from `aChooser`, allows to save, load and delete configuration presets for remote objects (3.2.2).

- `ThermalCycleChooser`, derived from `PresetManager`, enumerate previously saved thermal cycles allowing to load or delete them.

- `DeviceRole` connects to `*Role` and `*RoleIO` options, visualized by two or three combine boxes (combo) respectively. The first one lists all possible devices for the required role. Once the device has been selected, the second box lists all saved configuration presets for that device. If the option is `*RoleIO`, a third combo box allows to choose the option to be used as input/output.

The `aTable` widget represents editable tabular options, "`Table`"-type. It is composed by a set of custom widgets realizing a Model-View-Controller scheme, or MVC [Gamma et al., 1994]. The table data is actually contained in the `aTableModel` class, implementing all methods to access it in terms of rows and columns (the data Model). The `aTableView` class handles drawing the table to the user (the View), while the `aTablePointDelegate` defines how data should be edited (the Controller). Most of the MVC implementation is inherited from the Qt library [Project, 2012], respectively from `QAbstractTableModel`, `QTableView` and `QItemDelegate`.

Those widgets serves most of the Client functionality. Interfaces to configuration options are dynamically generated and organized by instantiating the proper `ActiveWidget`-derived class, based on the option type metadata.

4.3 Basic Plotting Facilities

Plotting is almost completely inherited from the Veusz scientific plotting package. As Veusz is well-written in pure Python, it is quite easy to take advantage of a basic subset of its capabilities in order to draw simple plots.

Two classes, `VeuszPlotWindow` and `VeuszPlot`, extracts from Veusz just the very essential plotting and data management functionalities, by inheriting, overriding and using (Figure 4.7):

- `PlotWindow`, where Veusz actually draws any plot;
- `CommandInterface`, receiving commands to add, remove or configure the plots;
- `CommandInterpreter`, allowing to control the plot with a simplified scripting language;
- `Document`, storing all data to be plotted and all metadata about objects constituting the plot.

The `VeuszPlot` class adds some useful feature, which would be pointless for the parent Veusz package, but necessary in Misura environment.

For example, it automatically adapt the plot to a container layout or parent widget. If the containing environment is under a guard dimension, the plot will automatically remove axis labels and ticks.

Figure 4.7: Plotting and Thermal Cycle Designer Class Diagram

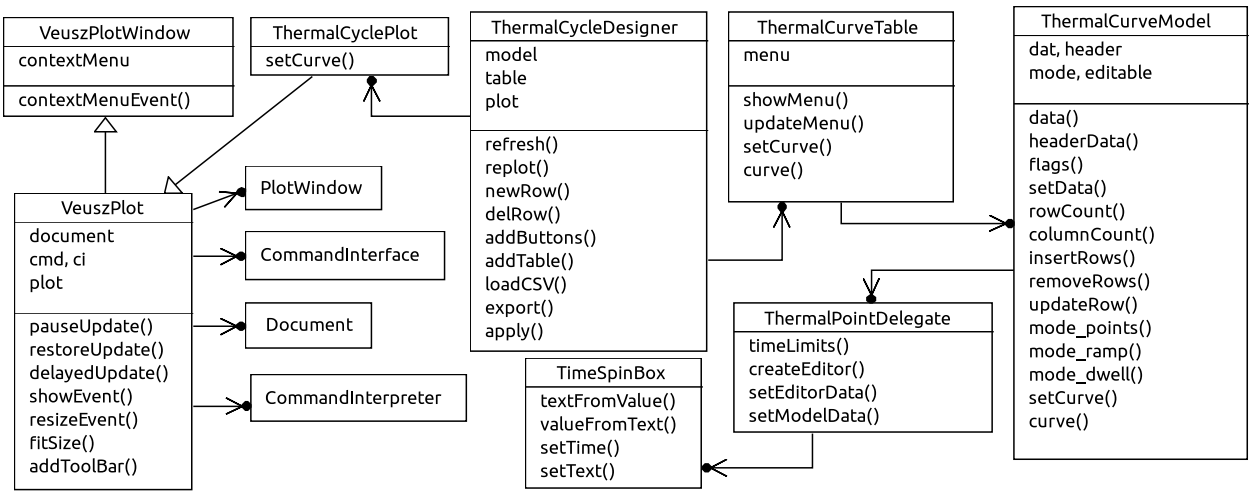
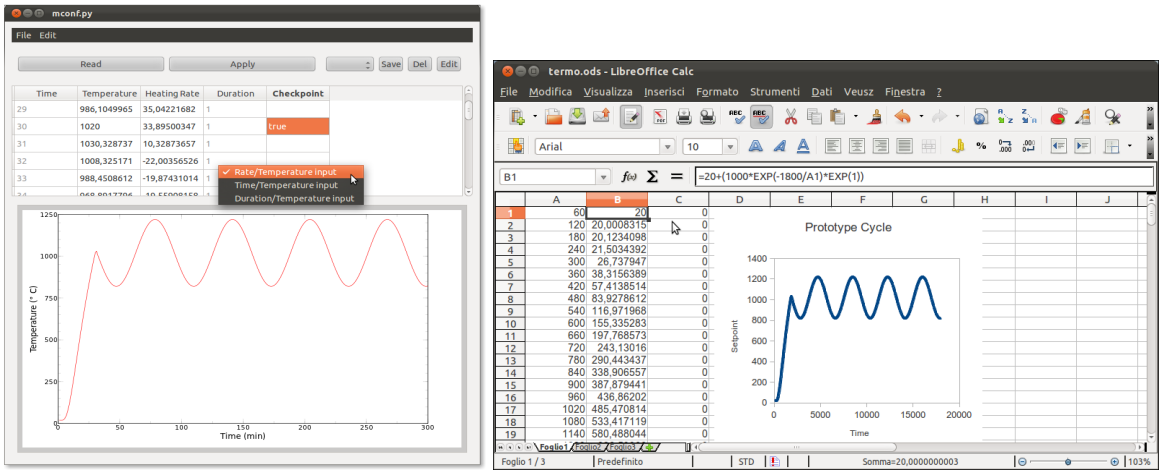


Figure 4.8: Thermal Cycle Designer (left). Standard spreadsheet tools (right).



4.3.1 Thermal Cycle Designer

The Thermal Cycle Designer is an example application of simple plotting. The interface comprises two main graphical elements (Figure 4.8):

- A table showing and allowing to edit the thermal cycle data
- A temperature plot as a function of time, for visualizing the resulting theoretical curve.

Moreover a set of menus and buttons allows to load/save/delete predefined cycles, and create or remove rows from the current one.

The tabular part of the interface, modeled in Class Diagram 4.7, follows the same architectural model of presented for the `aTable` active widget. The `ThermalCurveTable` class acts as a view to the data, modeled in `ThermalCurveModel` class. Upon cell double click, a `ThermalPointDelegate` class is instantiated presenting the proper editor. In the case of time column, the editor is a special `TimeSpinBox` instance.

Editing may happen in three different modes, configurable through a context menu:

- In *Rate/Temperature input* mode only those two columns may be edited. The user should decide up to which temperature the heating rate should be maintained for the corresponding entry. For example: "up to 800 at 20°C/min, then up to 1200°C at 5°C/min", would require two lines: T=800; R=20 / T=1200 ; R=5. This is the traditional way of working of Misura 3 and most dilatometers.
- *Duration/Temperature input* permits to specify how long the furnace should take to reach a determined temperature. This is usually needed in order to obtain dwells or stasis in the thermal cycle (where heating rate would be zero). Duration/Temperature notation may be more comprehensible to industrial kiln operators, where few thermocouples records the temperature in some critical points and only the time needed for the material to walk the distance between the two is known.
- In *Time/Temperature input* mode the user may directly specify the desired setpoint at any given time. For example, "after 50 minutes temperature should be 570°C" would require the line t=50; T=570.

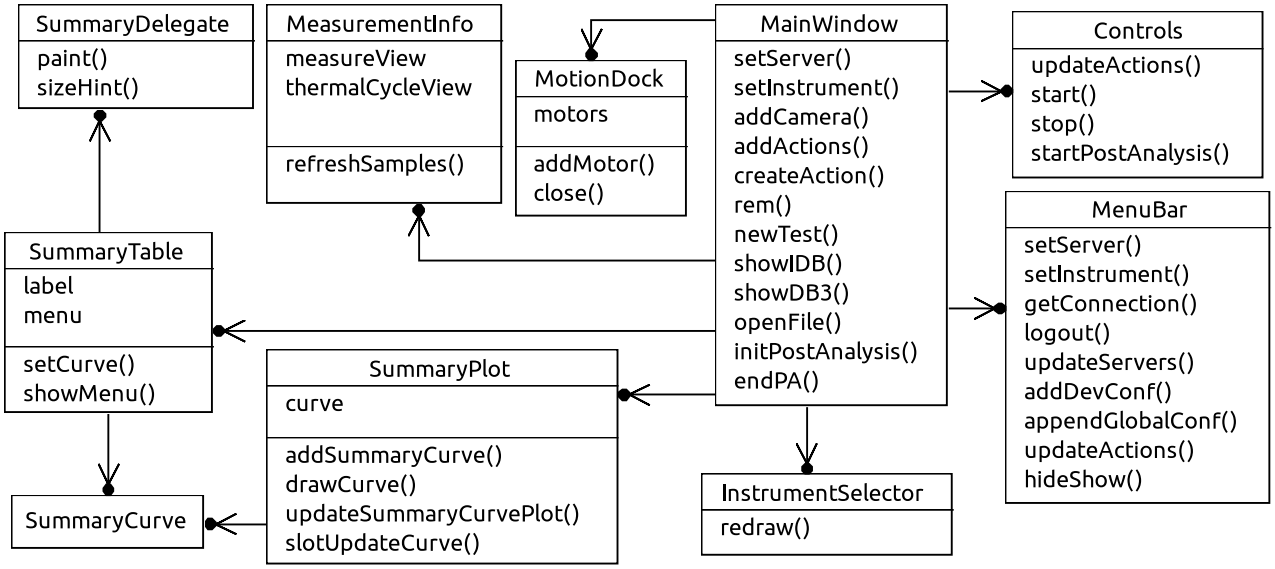
The `ThermalPointDelegate` will create a special checkbox when the user wants to edit the checkpoint column.

The designer can import and export Comma Separed Values (CSV) text files containing time, temperature and checkpoint data. This easily allows the operator to generate arbitrarily complex thermal cycles using the tools he is used to, spanning from spreadsheets up to advanced mathematical packages.

The Server thermal cycle validation procedure will automatically lower the heating rate if it exceeds the maximum speed of the furnace (ref. 6.1.2.2).

The lower part of the interface is occupied by a `VeuszPlot` object, with one curve and two axis. Every time a table cell is edited, the datum is refreshed in the Veusz

Figure 4.9: Acquisition Class Diagram



`Document` instance detailing all plot objects, and the curve accordingly changes. If the designer window is resized to a small size, axis, labels and ticks are removed and the curve enlarged.

4.4 Acquisition

The acquisition interface follows the general Client philosophy of *building upon Server specifications*. There is one single interface for all `Instrument` classes, comprising:

- general utility widgets, like the data plot and table, the log buffer, network connectivity dialogs, measurement and thermal cycle panel
- device-specific widgets, like camera views, motion controls, sample properties panels which are built based on the Instrument's configurations.

The classes constituting the acquisition GUI are modeled in Class Diagram on the current page.

InstrumentSelector This simple widget displays a list of available server and their capabilities.

SummaryPlot and Table `SummaryPlot` is responsible for the plotting of every remote summary table entry. `SummaryTable` displays the data in a tabular view, were acquired values are represented as rows and columns correspond to time values. During acquisition, new columns are appended as points are recorded.

If images are saved during acquisition, a table row will contain all saved frames.

Both widgets are built following the MVC principle and rely on the previously defined `SummaryCurve` data model (ref. 4.1.1).

MeasurementInfo Usually on the right of the main window, this *tabbed* object holds three configuration sub-panels:

- an interface to the remote Instrument.measure object, containing metadata about the test being executed.
- the thermal cycle designer
- one tab per each defined sample, holding sample metadata in multi-sample tests.

Motion Dock This panel contains controls for the motors configured in the current instrument, if any.

Controls The main toolbar under the menu. Usually it contains only Start and Stop buttons, which serve both for acquisition control and as a status indicator. In Post Analysis instrument the `Controls` toolbar expands for database access and test file loading options.

MenuBar The `MainWindow` menu is dynamically generated. Its entries allows the connection to machines and instruments, the configuration of the instrument or its devices and options over elements displayed in the interface.

MainWindow The main window combines together all former widgets, and coordinates their update upon connection to new servers and instruments (Figure 4.10). The interface is organized as a workspace, with panels holding data and configuration options, and a central space holding movable sub-windows for cameras and plots.

4.4.1 Camera interface

Cameras needs a special interface, first of all because they deal with images which must be timely displayed. Moreover, some of their options are better comprised and manipulated directly on the image bi-dimensional space, rather than by configuring obscure numbers on sliders and other Active Widgets.

Camera-related classes (also called `Beholder` GUI) are modeled in Class Diagram 4.11.

The speed issue is addressed by running a separate `QThread`, the `FrameProcessor`, which takes care of asking for the latest available frame on the Server and sending it to the `ViewerPicture`. The latter is a picture (Figure 4.12) with a context menu, from which it is possible to directly access to various configuration options and tools, like `BoxROI` and `CalibrationTool`.

Regions Of Interest The BoxROI class draws on top of the image one rectangle for each defined sample. In multi-sample instruments, each sample gets its own portion of sensor, and Misura must know *where* they actually are.

Figure 4.10: Acquistion Window

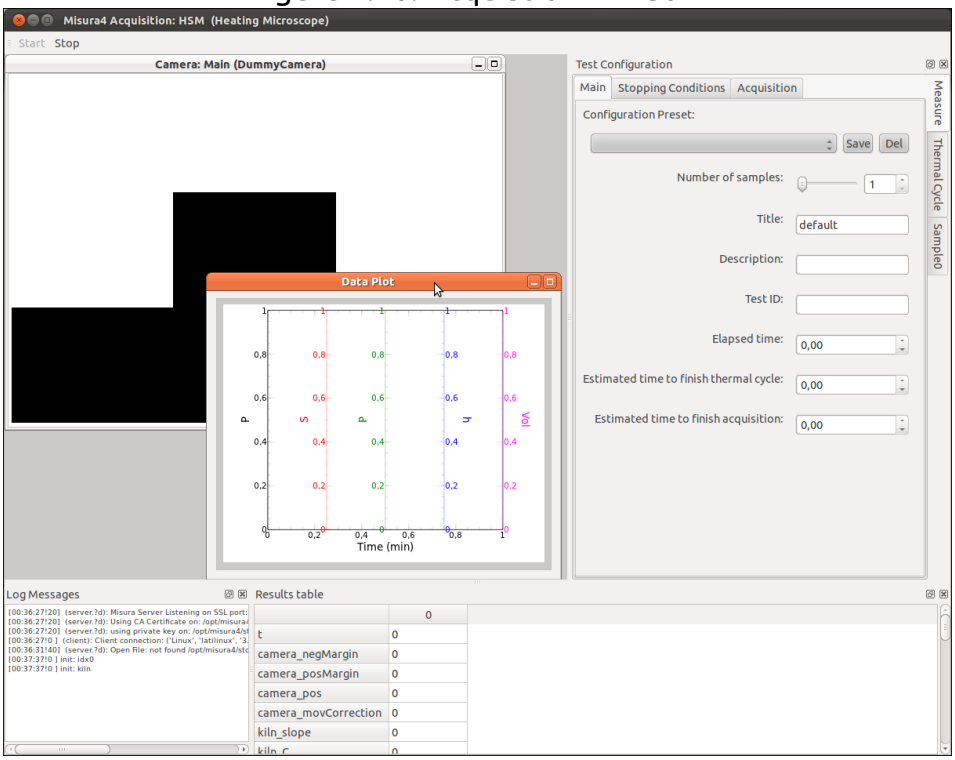


Figure 4.11: Beholder GUI Class Diagram

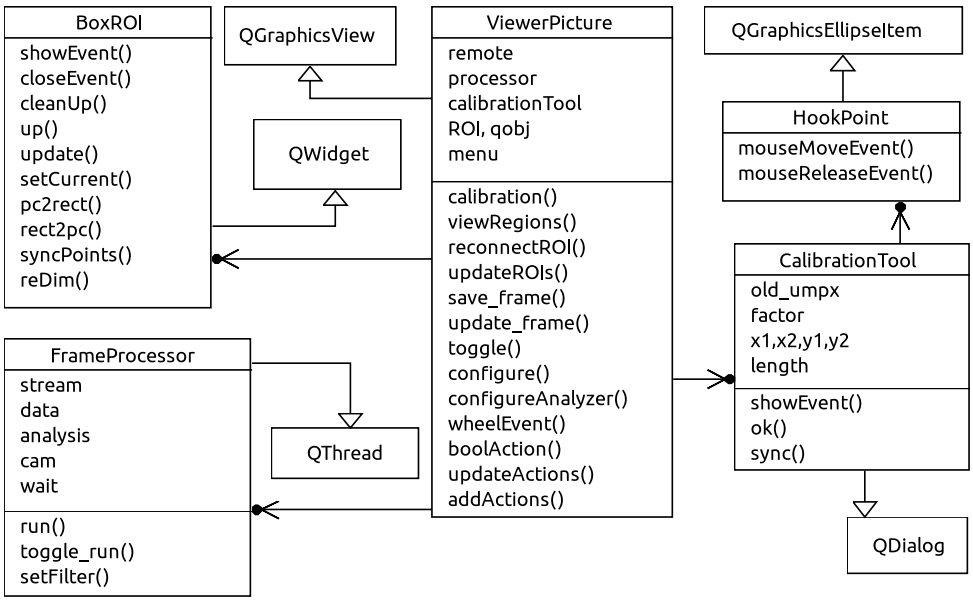
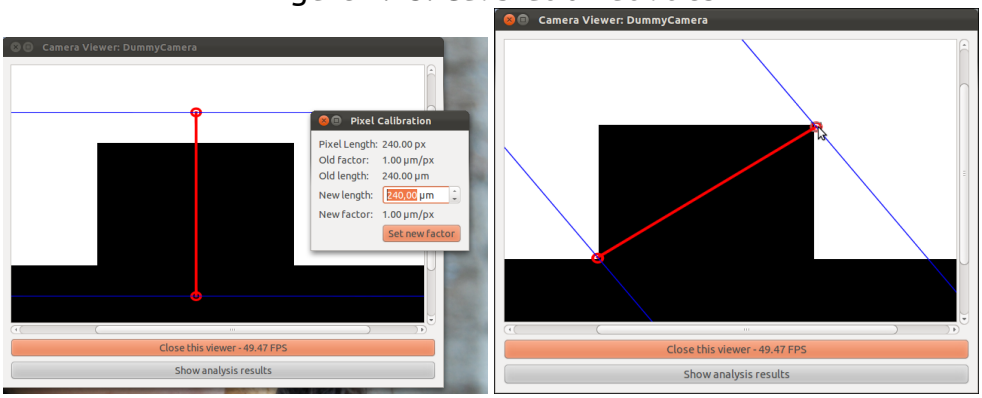


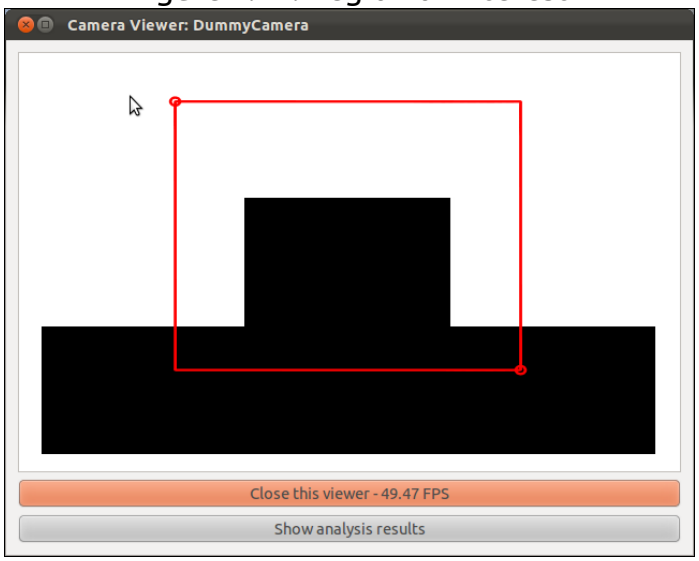
Figure 4.13: Calibration utilities



Sample position is contained inside a `Dict`-type option, holding x1, y1, x2, y2, opposite points defining a rectangle. Finding and expressing those rectangles by numbers would be cumbersome.

BoxROI rectangles allows simple positioning of both coordinates (Figure 4.12).

Figure 4.12: Region of Interest



Pixel Dimension Calibration Another delicate task is to define the viewed physical dimension corresponding to each image pixel. Indeed, the internal unit of measure of Misura is a pixel. Before calculated values are returned to the user or saved to the test file, they are converted into physical units (μm, microns).

The conversion factor is determined by placing an object of known dimension and measuring its pixel length. The left screen-shot in Figure 4.13 shows the `CalibrationTool` dialog. The red length between two points and blue parallel lines corresponds to 240 pixel, and the real length (in μm) is asked. That line can easily be moved in order to match any real object point-to-point distance (ideally, between parallel borders by helping with perpendicular blue lines).

Chapter 5

Image Analysis Methods

This Chapter contains the most exciting innovations introduced by Misura 4 in both optical dilatometry and heating microscopy.

These innovation were possible thanks to an advanced management of imaging devices, following the parallel model discussed in Chapter 2, and a rigorous organization of image analysis related code.

While basic computer vision algorithms are directly *imported* from the industry-standard OpenCV library (short name `cv`), many custom methods were implemented in order to extract as many information as possible from each single frame received from the camera.

Microscopy analysis was taken one big step forward, well beyond any conceived international standard, allowing the experimenter and the researcher to study new behavior and new materials.

5.1 Imaging devices

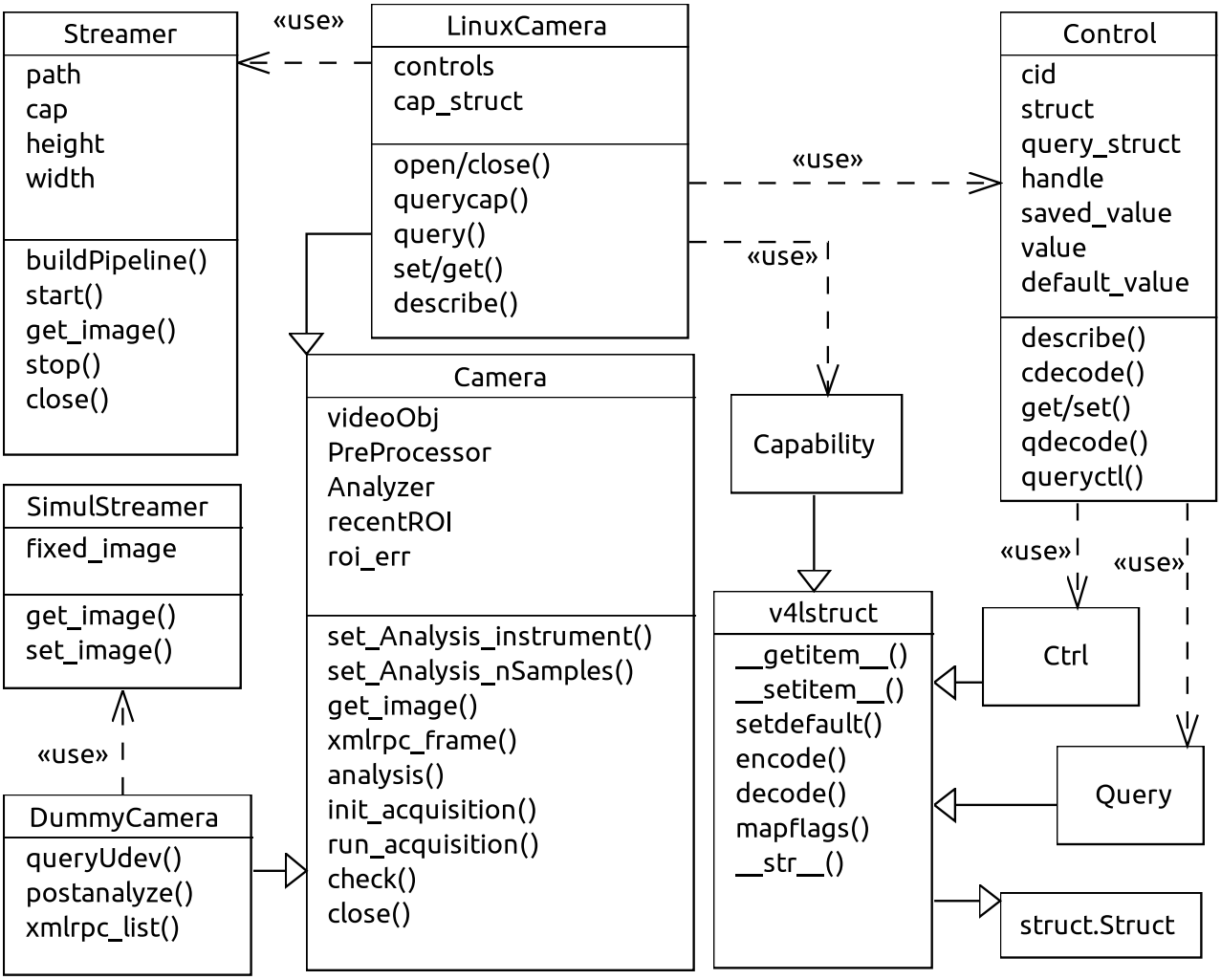
Misura has several levels of abstraction in order to simplify imaging tasks, modeled in Figure 5.1:

- An abstract `Camera` class, containing device-agnostic methods.
- A `LinuxCamera` class, dealing with real peripherals in collaboration with a `Streamer` class grabbing frames.
- Operative system low-level classes `Capability`, `Control`, `Ctrl`, `Query`, `v4lstruct` for device configuration.
- A `DummyCamera` virtual device class, getting images from a `SimulStream` frame simulator.

5.1.1 Connecting to camera devices

Misura takes advantage of two technologies in order to control imaging devices: the Video 4 Linux 2 (V4L2) application programming interface [LinuxTv, 2011], included

Figure 5.1: Imaging devices Class Diagram



in the standard linux kernel, and the GStreamer open source multimedia framework [GStreamer, 2012].

The V4L2 API allows Misura to query devices for their capabilities and available controls, such as Brightness, Contrast, etc. Through simple read/write and I/O-CTL (ioctl) requests, the device settings can be obtained and modified. I/O-CTL is a system call for device input/output operations, extensively used by the V4L2 API as a mean for controlling devices.

The GStreamer framework allows to simply setup and manage an image stream with many kinds of devices. Its well structured architecture allows to pass the stream through a *pipeline* of subsequent plugins which, for example, reduce the image to gray-scale and convert it to a standard format (JPEG).

These components concur in the whole functionality of the Misura representation of a camera device, as modeled in Figure 5.1.

The `DeviceServer` managing imaging devices, `Beholder`, instantiates a `LinuxCamera` for each `video<N>` device found in the `/dev` directory, plus a `DummyCamera` object used for testing and post-analysis of static images. Both classes inherits the abstract `Camera` object, containing general acquisition and analysis related methods and attributes.

Upon V4L2 device initialization, its corresponding `LinuxCamera` instance will iterate over all possible ioctl in order to find supported controls. Calls are made using binary structures represented by the general `v4lstruct` object:

- `Capability` structure contains the binary ioctl request for asking and reading which acquisition modes are supported by the device: read/write or memory mapping.
- `Control` structure represent a device control, like Brightness or Saturation, by grouping together Query and Ctrl structures.
- `Query` asks and reads the current value of a control.
- `Ctrl` sets a control to a new value and reads the status reply.

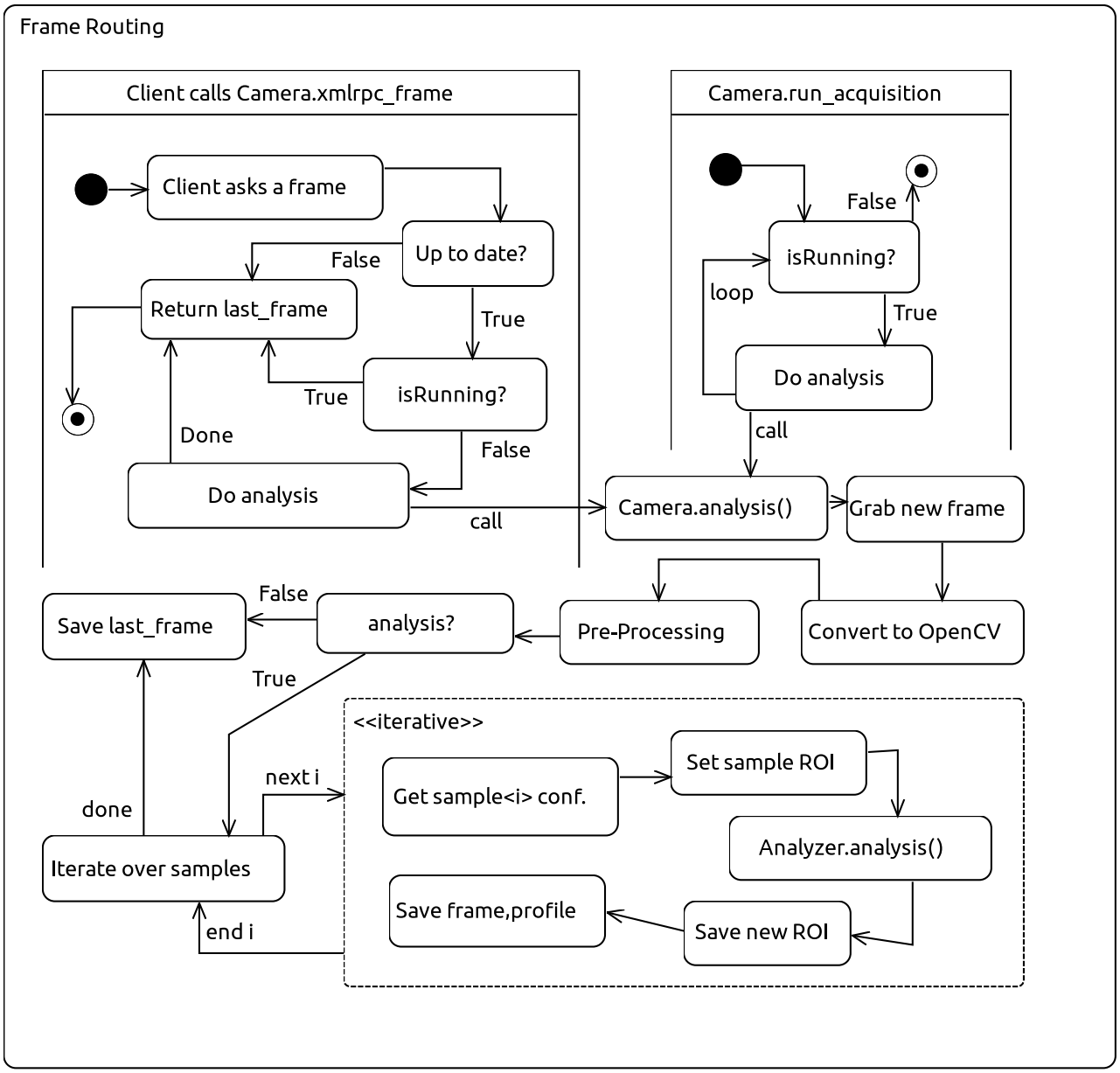
Camera settings are stored inside `LinuxCamera` options, whose value is related to the hardware value by overridden `get/set` methods.

5.1.2 Abstract `Camera` functionalities

The abstract `Camera` class has two main purposes: properly *routing* grabbed frames across pre-processing, analysis, and the Client, and realizing the acquisition loop, modeled in Figure 5.2. Those tasks are totally independent from the underlying device: they are shared between both `LinuxCamera` and `DummyCamera` child classes.

Frame routing deserves further discussion. When the Client asks for a frame, usually for user visualization, it attaches to the request the sequence number of the last received frame. If the current frame sequence number is bigger than Client's request, the in-memory frame (saved in the `last_frame` option) is returned: no frame is really grabbed from the device. This way if two concurrent Clients has simultaneous

Figure 5.2: Frame routing Activity Diagram



connections, only the quickest one determines real hardware workload, while the slower one will just read the newest in-memory frame. This behavior realizes *intra-Client prioritizing*, based on request times and thus, speed.

Another prioritizing takes place between Clients and acquisition requests. If the `run_acquisition` process of the camera is running, no new frame can ever be grabbed as a consequence of a Client request. Acquisition totally blocks Client frame requests, by only returning newest in-memory frame. This *total acquisition priority* avoids that the client can someway slow down the acquisition process.

If the Client is entitled to receive a new frame, the analysis method is called. Here the frame is grabbed from the device, converted to a proper OpenCV data structure and preprocessed according to camera settings (see 5.2.1). If the actual analysis is not requested, the frame is saved as `last_frame` and sent to the client without any further elaboration: only preprocessing is applied.

If the analysis is requested, each sample defined on the active Instrument is iteratively retrieved and passed as argument to the current `Analyzer.analysis` method. `Flex, Odlt, Odht` instruments only have one possible sample, so the iteration runs just once. But `Hsm, PostAnalysis` and `Drop` instruments may define up to 8 samples, each confined in a rectangular Region of Interest (ROI) inside each single frame, defined as an option holding a rectangle coordinates (x, y, height, width).

The real analysis is performed only inside those regions, and results are stored inside the sample configuration together with the ROI frame piece and the detected profile.

5.2 Analysis overview

Actual image preprocessing and analysis is performed in the `analysis` module, modeled in Class Diagram 5.3.

Analyzers are defined as pure Python `object` classes, without any configuration, network, or multiprocessing feature. Each `Camera` instantiates two Analyzer-derived classes as attributes, `Camera.PreProcessor` and `Camera.Analyzer`, and uses them during the image routing in order to perform preprocessing and analysis tasks.

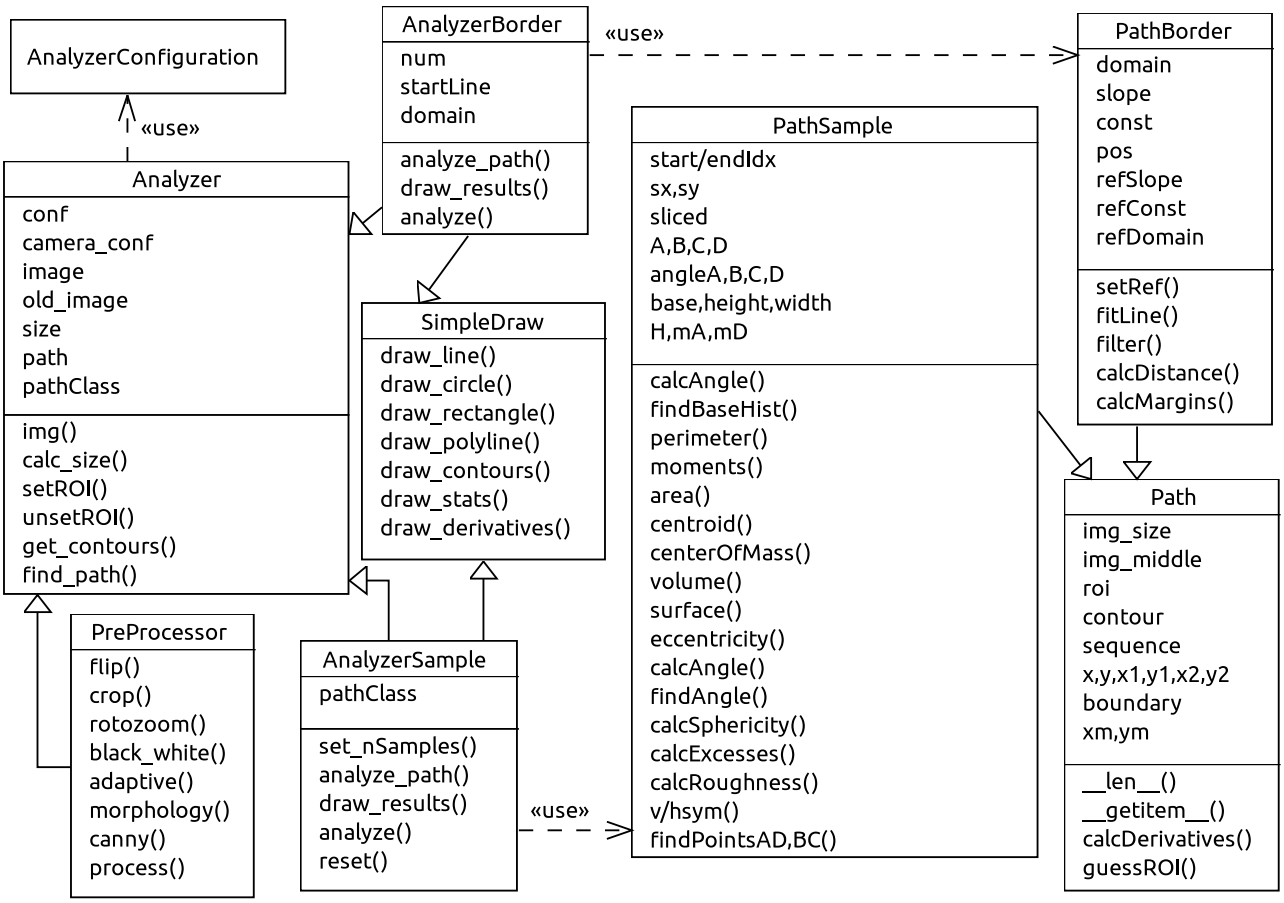
Since `Analyzer`-derived objects does not contain any configuration, they access to options contained inside their `Camera` owner, referenced in the attribute `camera_conf`, and in the global instrument configuration, referenced in attribute `conf`. Those attributes are set during initialization of the class.

A `Camera` may anytime change the instrument for which it is doing the analysis (for example, the device may be opened by the `Hsm`, then by the `Drop` instruments). In that case, the `Camera.set_Analysis_instrument` method will destroy the obsolete `Analyzer` instance, and instantiate a new object according to the selected instrument.

Some simple Analyzer methods are:

- `img()`, checks and imports an OpenCV image.

Figure 5.3: Analysis Class Diagram



- `calc_size()`, calculate the size of the image and updates various internal metadata.

- `setROI(), unsetROI()`, tells OpenCV to limit all operations to a sub-rectangle of the image, the Region Of Interest.

The main task of `Analyzer` is to find the sample, without caring if it is an Hsm whole sample or a dilatometer sample border. This is accomplished by following the assumption that *the sample shadow must be the protagonist of the frame* or frame region: the biggest dark object found is identified as the sample. If that is not true, the analysis cannot succeed: the user must find a way to let the sample be the biggest and darkest thing on the scene.

The OpenCV library offers a practical function to find all *contours* in binary images, `cv.FindContours`, called in the `get_contours()` method. A contour is the perimeter of a region of uniform pixels (all-black or all-white). The algorithm is based on a Suzuki work [Suzuki and Abe, 1985] , which allows to also distinguish the containment relations between contours.

Since the protagonist assumption is accepted, in `find_path()` the contours are sorted according to their length, and the longest one is identified as the sample and memorized in a `Path` object. A `Path` holds a sequence of (x,y) pixel coordinates drawing a curve on the image, and defines methods for manipulation and characterization of that curve.

Although there are many defined instruments, the effective kinds of image analysis are reduced to three `Analyzer`-derived classes:

- `AnalysisBorder`, used by `Flex, Odlt, Odht` cameras

- `AnalysisSample`,used by `Hsm` and `PostAnalysis` cameras

- `AnalysisDrop` for the `Drop` instrument, not discussed this dissertation as in early stage of development.

The differences between these classes are due to the way they manipulate the sample profile, respectively represented by two `Path`-derived class: `PathBorder` and `PathSample`.

Finally, the `SimpleDraw` class contains methods used for drawing shapes and curves onto an image file, particularly useful for debugging. They mainly are simplified shortcuts to OpenCV functions with similar names:

- `draw_line` calls `cv.Line`

- `draw_circle` calls `cv.Circle`

- `draw_rectangle` calls `cv.Rectangle`

- `draw_polyline` calls `cv.PolyLine`

- `draw_contours` calls `cv.DrawContours`

- `draw_stats` writes some diagnostic text, using `cv.PutText`
- `draw_derivatives` display curves referring to the detected sample profile

5.2.1 Image Pre Processing

The imaging device usually returns a gray-scale image, which is further manipulated in order to emphasize its features and minimize signal to noise ratio. This phase is called Pre Processing, as it precedes the shape detection and analysis. The `PreProcessor` class implements general utility methods, useful for any kind of subsequent analysis. Most of the functionality is imported from the `cv` module.

Since their application highly depends on which device the images are retrieved from, as they help correcting lighting conditions, dust, etc, the `PreProcessor` configuration options reside inside the `Camera` device which owns its instances, as explained in 5.2.

Each `PreProcessor` method is triggered by a `Camera` option, starting with `Pre-Processing:` string. Implemented methods are:

`flip()` control option `:Pre-Processing:Flip`

Flipping is natively supported by many devices, anyway it is made available to anyone thanks to the flip method. The image can be symmetrically turned by 180° around its vertical or horizontal axis - or both.

Based on `cv.Flip` function.

`crop()` control options `:Crop, :CropRect`

This method will crop the image at the desired dimension. Cropping is governed by a boolean option, and a dictionary option holding the x, y, height, width coordinates of the target rectangle.

Based on `cv.GetSubRect` function.

`rotozoom()` control options `:Rotation, :Zoom, :ZoomCenter`

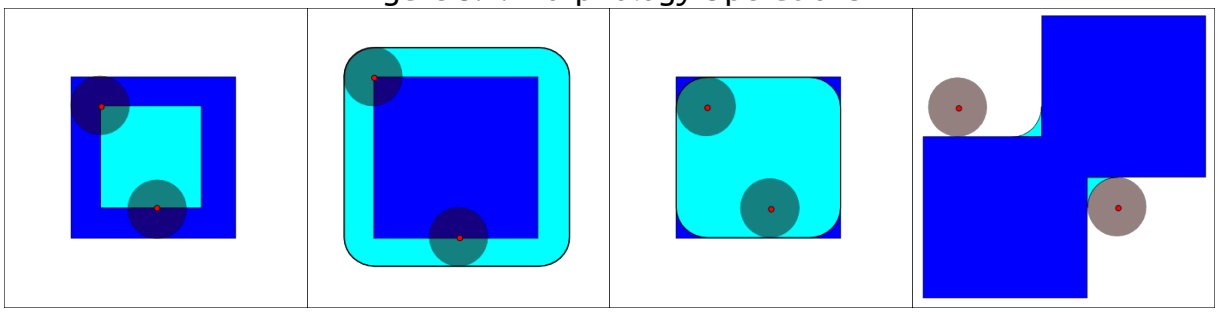
Rotation and zooming are handled together by using sub-pixel accuracy. The output image is determined by manipulating each source pixel using a transformation matrix. The transformation matrix is built using the zoom factor f (0-1), the rotation angle r, and the center of zooming coordinates cx, cy (normalized by the source image size).

$$cx = \frac{zoomCenterX}{srcWidth}\ ,\ cy = \frac{zoomCenterY}{srcHeight}$$

$$M = \begin{bmatrix} f\cos(r) & f\sin(r) & cx \\ -f\sin(r) & f\cos(r) & cy \end{bmatrix}$$

$$out(x,y) = src(M_{11}x' + M_{12}y' + M_{13}\ , M_{21}x' + M_{22}y' + M_{23})$$

Figure 5.4: Morphology Operations



From the left: Erosion, Dilation, Opening, Closing. Source: Wikipedia.

$$x' = x - \frac{outWidth - 1}{2} \; , \; y' = y - \frac{outHeight - 1}{2}$$

Based on `cv.GetQuadrangleSubPix` function.

`morphology()` control options `:MorphologyOpening, :MorphologyClosing`

Opening and Closing are mathematical morphology operators, resulted by combining two more operators - Erosion and Dilation [Serra, 1983]. Applied to image processing, a small *structuring element* or *kernel* is swept over the entire image and, based on how it fits underlying region, it is decided how to transform it. Usually, that structuring element is a small rectangle or circle, having an *anchor point* at the center.

In the case of Dilation, the source pixel under the anchor point will be set to the maximum pixel value found across the kernel, causing bright regions of the image to grow. Erosion does the opposite, setting the source pixel under the anchor point to the minimum pixel value in the kernel.

The Opening operation will first erode than dilate the image, obtaining that isolated dark regions are filtered out. The Closing first dilate then erode, obtaining that isolated bright regions are reduced.

Effects of these operations are exemplified in Figure 5.4.

The control options store the rectangular kernel width,height and the number of iterations for the morphologic opening/closing operations: if set to zero, no operations will be done.

Based on `cv.MorphologyEx` function.

`black_white()` control options `:BlackWhite, :white, :black`

Contour detection algorithms work better on black and white images. This method will apply a threshold to pixel values: all pixel less than `:black` will be set to zero (black), while all pixels more than `:white` will be set to 255 (white).

Based on `cv.Threshold, cv.InRangeS, cv.Set.`

`adaptive()` control options `:AdaptiveThreshold, :blockSize, :detract`

Adaptive thresholding acts following a similar principle as `black_white()`, except that the threshold levels (former `:black, :white`) are variable [Jain, 1989]. The

threshold is calculated independently for each pixel, by considering its neighbor pixels (:`blockSize`) and by subtracting a constant (:`detract`).

The decision whenever to black-out or white-in a pixel depends on the neighborhood of that pixel. For example, a slightly gray pixel found in a perfectly white region will be blacked to 0, while a not-so-dark pixel found in a totally black region will be whitened to 255. This way gradients are greatly enhanced and the resulting thresholded image is way less sensible, for example, to lighting conditions.

Based on `cv.AdaptiveThreshold`.

5.2.2 Whole Path techniques

The great advancement in analysis quality, flexibility and reliability obtained in Misura 4 is achieved thanks to *whole path techniques* for sample characterization. The concept is that every single contour pixel is relevant and precious, and should be taken into account when calculating material properties.

The `Path` class is the basis upon complex characterizations are built. It is initialized with an OpenCV sequence structure as input, which is converted to NumPy arrays of x and y point coordinates. Knowing original `img_size` image dimension, a `boundary` condition is applied in order to filter out all pixels near the sides of the image, and reconstruct the contour by filling consequent gaps. New, filtered coordinates are then stored as `Path.x, Path.y` array attributes, along with `xm,ym` mean points.

The `calcDerivatives()` method calculates first and second order derivatives with respect to the index of a point in the curve, using a discrete derivative function D:

$$\begin{aligned} x' = D(x,j)\ ,\ y' = D(y,j) \\ x'' = D(x',j)\ ,\ y'' = D(y',j) \end{aligned} \tag{5.1}$$

The discrete derivation function D takes an input parameter, j, which define the index difference to be used in the derivation. The bigger j is, the farer the comparison point is searched back in the curve index. The function actually works by creating two copies of the x array, P and A. The A array is appended with j new elements equal to the last curve element, x_n, while the P array is *prepended* with j copies of the first x_1 element, as showed in Equation 5.2.

$$\begin{aligned} x &= [x_1 \quad x_2 \quad ... \quad x_n] \\ A &= [x_1 \quad x_2 \quad ... \quad x_n \quad x_{n+1} \quad x_{n+2} \quad ... \quad x_{n+j}] \quad (x_{i>n} = x_n) \\ P &= [x_{1-j} \quad x_{2-j} \quad ... \quad x_0 \quad x_1 \quad x_2 \quad ... \quad x_n] \ (x_{i<1} = x_1) \\ D_k &= \frac{P_k - A_k}{j} \ (j < k < j+n) \end{aligned} \tag{5.2}$$

Results are stored as `Path.x1, x2, y1, y2` array attributes.

The `guessROI()` method will identify the optimal region of interest for the *next*

image analysis, by considering the position of the sample profile within the total image size. If the sample moves or changes dimension, the algorithm will be able to follow it by keeping the region of interest at a security distance from the detected profile. This way it is possible to analyze many samples at the same time: by allowing the analysis to be performed inside the smallest possible region around the sample, the risk of one sample invading the region occupied by other samples is greatly reduced.

5.3 Border Analysis for Dilatometry and Fleximetry

Vertical and Horizontal Dilatometers and Fleximeter are based on the idea of independently following the smallest possible region on the border of sample, and then adding the observed movements in order to find the total expansion, contraction or flexure. Dilatometers will follow two sides of the same sample and combine the results into a final expansion value, while the Fleximeter just uses one camera looking the sample in its median point.

The image analysis task associated with these instruments is identical: identify the position of a border, which cuts the image in a black region and a white region. It may seem the simplest computer vision effort, and for this reason it comes first in the present dissertation. The complexity increases when border detection is associated to high-precision instruments and noisy signal due to unavoidable border imperfections found in every material, sample cutting difficulties, plus thermal effects changing sample's shape *during the analysis*.

It is not possible to eliminate all these sources of errors by writing advanced algorithms, but it is possible to mitigate them to some extent and in many adverse conditions.

Figure 5.5 models the activity flow involved in a border analysis. Before the real analysis begins, an initialization procedure executes a line fitting on the path and store cumulate results.

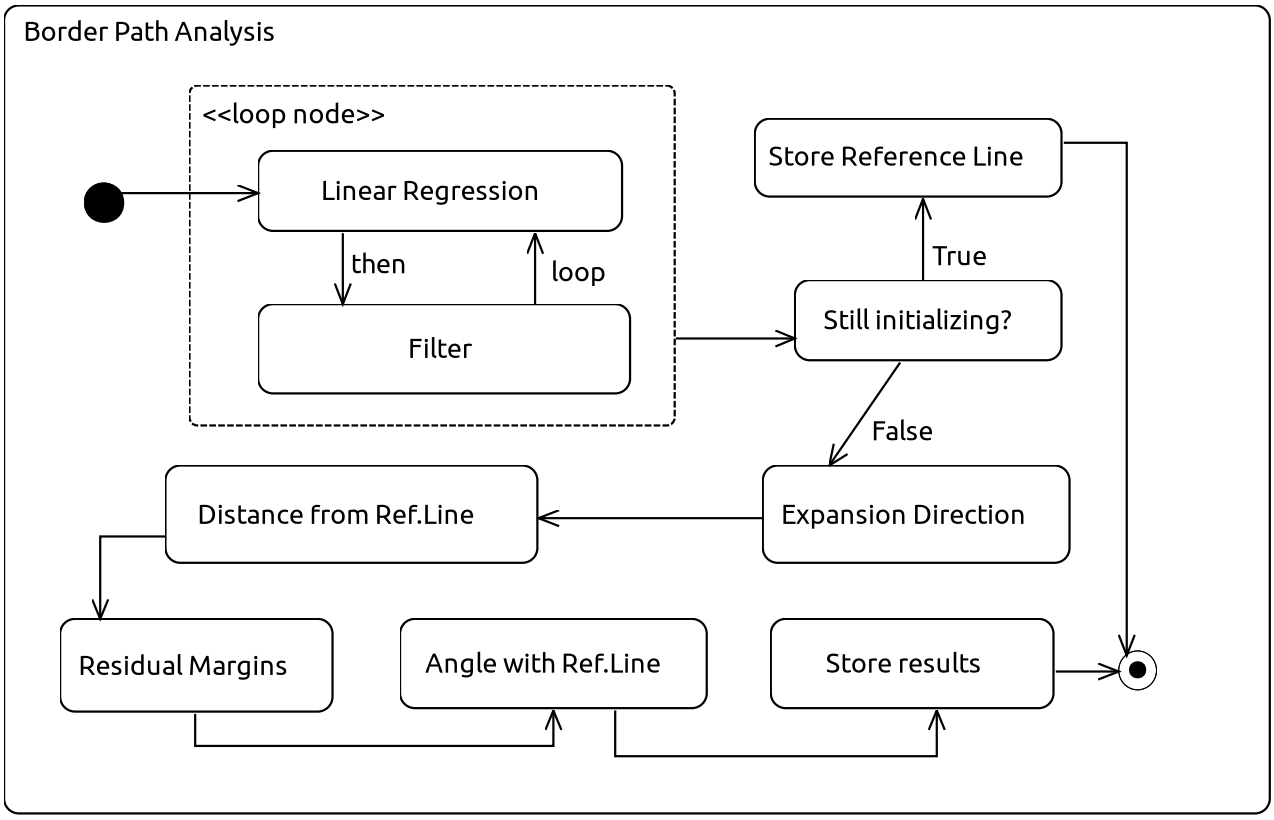
The `PathBorder.fitLine()` method performs a linear regression fitting on the x,y points representing the path.

If the line fitting has a coefficient absolute value lower than one, it means the path is dividing the frame in upper and lower regions. Instead, if the coefficient absolute value is bigger than one it means the paths divides the frame in left and right regions. In this case, fitLine automatically invert the domain of the curve by switching x, y arrays, in order to always work in an up/down-regions condition.

If specified, the path may be filtered from all points which are too far away from the fitted line, like peaks or valleys, and the fitting be repeated. After the configured number of fit-filter iteration, slope, constant and error values are stored and returned to the caller.

After the initialization procedure is complete, a *reference line* is built by averaging slope and constant values from results cumulated by previous iterations. This will be the zero towards which all future borders will be compared in order to calculate their movement from the start of the analysis.

Figure 5.5: Border Analysis Activity Diagram



Once the current border is properly fitted and there is a reference line towards which to calculate the distance, the sign of that distance must be determined.

Is an up movement positive or negative? Is the sample expanding or contracting? Is it going up or down? It depends on where the sample is in the frame. If the sample is down, than an up movement means expansion, or positive flexure, and the path-reference distance must have a positive sign. If the sample is above the path, up movement means contraction, or negative flexure.

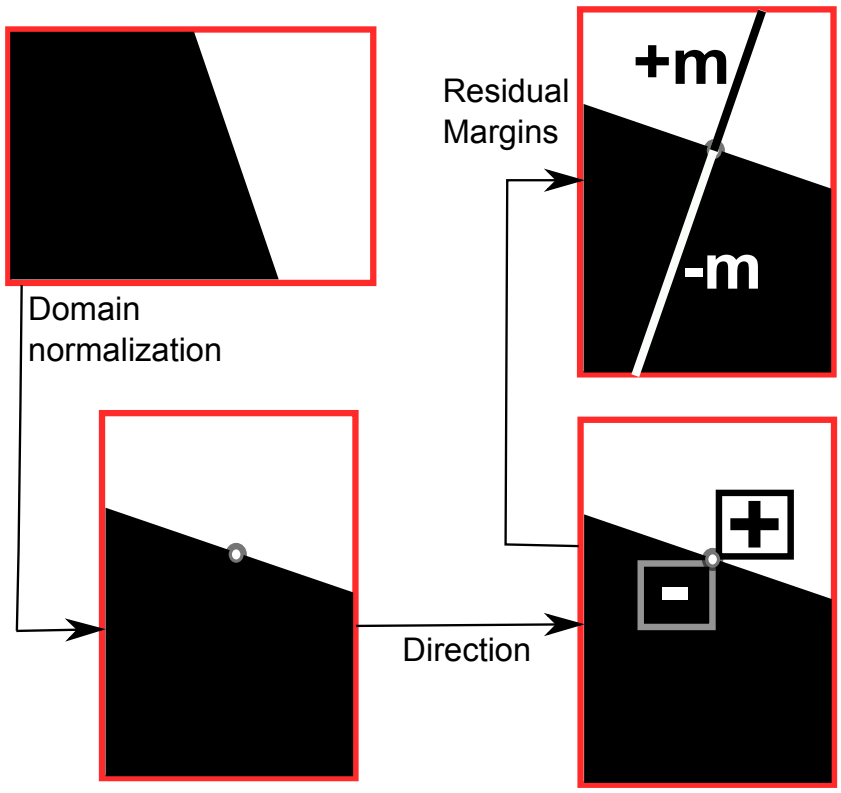
As showed in Figure 5.6, the sample position (and thus displacement direction) is determined by taking two rectangles at the opposite sides of the border. They share one vertex in the border median point which is also their respective center of symmetry, and are referred as Up and Down rectangles. All pixels contained in those rectangles are summed up: if Up>Down, than Up is the positive direction and Down is the negative. Otherwise, if Up<Down, the direction is inverted.

For example, if we imagine to draw the reference line on the left of the starting sample in Figure 5.6, in the dark region, the algorithm would calculate a positive displacement.

The `calcDistance()` method calculates the displacement by averaging the point-to-line distance of border points towards the reference line. The standard deviation of point-to-line distances across all border points is also saved as the error associated to the measurement.

Finally, the angle between the fitted line and the reference line is also saved as an error indicator, as when this angle changes too much during an analysis, the result is

Figure 5.6: Border analysis concepts



probably meaningless. Angle growth will probably be observed, for example, when the sample starts to soften and melt.

Motion Control Automatic motion control aims to keep the border inside the camera visual by moving the sensor when the sample is about go out of sight.

To help in this task, two more values are calculated in the `calcMargins()` method: positive and negative residual margins, defining the current border position relative to the ends of the space framed by the camera. First, it is found the equation of the line perpendicular to the border and passing through its median point. That line is then intercepted with image borders, and distance from those intercepts and the median point is calculated. The portion of distance residing in the positive region is called positive margin, while the other segment is the negative margin.

When one of the margins goes below a threshold value, motion control procedures will move the camera to the opposite direction, in order to restore safety margins.

This image analysis technique is tolerant towards profile imperfections as holes or peaks, inclination - also during the analysis, and correctly determines real motion margins.

5.4 Shape Analysis for Heating Microscopy

International standards set procedures and parameters in order to prepare a sample of material and identify characteristic temperatures by judging how the shadow of

that sample modify during an heat treatment. Usually, an operator would identify those shapes by eye or gross evaluations of simple geometric dimensions like height and width.

The advantage of demanding shape analysis to a computer vision algorithm, capable of automatically recognizing characteristic shapes, are:

- Increased reproducibility of the results, if recognition is based on robust mathematical properties of the shape itself.
- Automated working of the instrument, with no operator intervention needed, once the sample is correctly prepared.
- Adaptive behavior of the measurement. Temperature can be controlled depending on the characteristic temperature reached.

Seven standards were studied, and sample parameters considered for determining useful dimensional factor to be calculated in order to attain algorithmic recognition of shapes. Table 5.1 details the initial shapes required by these standards, listed by international organization abbreviation:

1. ISO, International Standards Organization, : Solid Mineral Fuels - Determination of fusibility of ash - High temperature tube method. {*ISO 540 1995-03*}
2. DIN, Deutsches Institut für NormungDetermination: of Fusibility of Fuel Ash. {*DIN 51730 1998-04*}
3. ASTM, American Society for Testing and Materials: Fusibility of Coal and Coke Ash. {*ASTM D1857-68 (Reapproved 1980)*}
4. CEN, European Committee for Standardization:
   (a) CEN/TS, Solid Biofuels - Method for the determination of ash melting behaviour - Part 1: Characteristic temperatures method. {*CEN/TS 15370-1: 2006-09*}
   (b) CEN/TR, Solid recovered fuels - Methods for the determination of ash melting behaviour by using characteristic temperatures. {*CEN/TR 15404: 2010*}
5. BS, British Standards: Methods for the Analysis and Testing of coal and coke. Part 15: Fusibility of coal ash and coke ash. {*B.S. 1016: Part 15: 1960*}
6. IS, Indian Standards: Methods of determination of fusibility of ash of coal, coke and lignite. {*IS 12891 : 1990*}

Each standard also defines its own criteria for characteristic shape detection, detailed in Table 5.2.

After having reviewed these seven Standards, it can be stated that they are not computer-friendly at all, as they all rely on operator judgment and personal experience in determining all of the shapes.

Table 5.1: Initial sample shapes

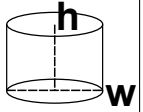
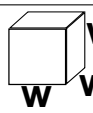
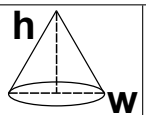
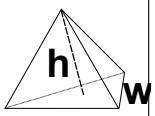
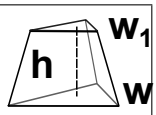
Units: mm	Cylinder	Cube	Cone	Pyramid	Trunk Cone	Trunk Pyramid
Standard						
ISO	$3 < h = w < 9$	$3 < w < 9$	-	$h < 19$, $\frac{h}{3} < w < \frac{h}{2}$	$h = 4$, $w = 3$, $w_1 = 1.5$	-
DIN	$h = w = 3$	$w = 3$	-	-	$h = 4$, $w = 3$, $w_1 = 1.5$	-
ASTM	-	-	$h = 19$, $w = 6.4$	-	-	-
CEN/TS	$3 < h = w < 5$	-	-	-	-	-
CEN/TR	-	$3 < w < 5$	$3 < h, w < 5$	$3 < h, w < 5$	-	$3 < h, w, w_1 < 5$
BS	$h = w = 3$	$w = 3$	-	-	-	-
IS	$2 < h = w < 3$	-	-	-	-	-

Table 5.2: Characteristic shape recognition as specified in standards

Code	Temperature	**ISO**	**DIN**	**ASTM**	**CEN/TS**	**CEN/TR**	**BS**	**IS**
TI	Shrinkage, Sintering	-	-	First rounding of cone vertex.	$0.95\ G_0$ area	$0.95\ G_0$ area	-	-
TD	Softening, Deformation	First rounding.	First rounding.	$h_D = w_D$ spherical shape	First rounding. $1.15\ \psi_0$	First rounding.	First rounding.	First rounding.
TS	Sphere	Pyramid: $h_S = w_S$ Cube, Cylinder: constant h, final rounding	$h_S = w_S$, final rounding	-	-	-	-	-
TH	Half-Sphere	$h_H = \frac{w_H}{2}$	$h_H = \frac{w_H}{2}$	$h_D = \frac{w_H}{2}$	$h_H = \frac{w_H}{2}$	$h_H = \frac{w_H}{2}$	$h_H = \frac{w_H}{2}$	Half-sphere.
TF	Fusion	$h_F = \frac{w_H}{3}$	$h_F = \frac{w_H}{3}$	$h_F = 1.6$	$h_F = \frac{h_H}{2}$	$h_F = \frac{w_H}{2}$	$h_F = \frac{w_H}{3}$	Flows.

The softening point is the most difficult to objectively determine: "first signs of rounding" is a quite vague concept, along with "sphere-like shape" or "completely rounded shape".

The only exception is the CEN standard, which specifically introduces a shape difference factor, F. It is the ratio between the perimeter of a theoretical circle of the same area as the sample, and the sample real perimeter (here referred as *roundness* property, see 5.4.2.2).

Misura 4 aims to generalize and automate shape characterization, by calculating as many shape properties as possible and correlating them to characteristic shapes (see characterization, 5.4.3). Simple properties like width, height, area G or roundness ψ are obviously available for users who wish to manually apply standard's specifications, and the image analyzer can be configured in order to strictly attain to the standard.

5.4.1 Fundamental Analysis Concepts

Whole profile techniques really shows their potential in sample analysis.

The following assumptions are made about how the sample is made and framed in the camera:

1. Both sample and sample holding plate (base) are framed by the camera, on both sides of the sample.

2. The plate pixel length is at least 20% of the framed profile length, at least 10% at each side of the sample.

3. The plate angle towards frame x axis should not be bigger than 89°.

4. Sample and plate does not have considerable convex regions. A star-like shape is not admissible, for example.

5. The angle between plate and sample should be the biggest angle in the whole profile near or in the plate.

6. For 3D features to be correctly calculated, the real sample should have a vertical axis of symmetry (cylinders and cone are good, pyramids and cubes are not).

The analysis is not limited to any specific shape, until it satisfies the assumptions. Due to assumption 6, best shapes are cylinders or cones, no matter which is their height/width ratio. Even rotation of the plate is admissible up to physically acceptable angles.

Figure 5.7 summarize the main preparatory steps in order to normalize the shape and find the critical points which distinguish the plate from the sample, called A, D, respectively the left contact point and the right contact point.

5.4.1.1 Pre-Rotation

After the dominant contour is found and boundary points filtered out according to general procedure described in 5.2, the profile is normalized against rotation towards frame X axis.

Rotation is usually an unwanted phenomenon, which may be observed because of an operator error or an optical system defect (the camera lost its correct angle). But in some cases it may represent a measurement parameter purposely set by the operator, e.g. to observe gravity influence on shape deformation with temperature.

The `PathBorder.preRotation()` method considers the first and the latest 10% of points in the path, uses them for a linear regression, and calculate the angle between the fitted line R and the x axis (Figure 5.7:c).

That **pre-rotation angle** θ is then used for a central rotation around the mean point of the whole profile. The x,y matrix is first translated by the centroid c_x, c_y coordinates, letting the center be the origin of the new translated vector (M'). Then it is rotated by multiplying the x, y matrix by a rotation matrix, depending on the rotation angle a (M''), and translated back to the original coordinates (M'''), as formalized in Equation 5.3.

Figure 5.7: Hsm analysis concepts

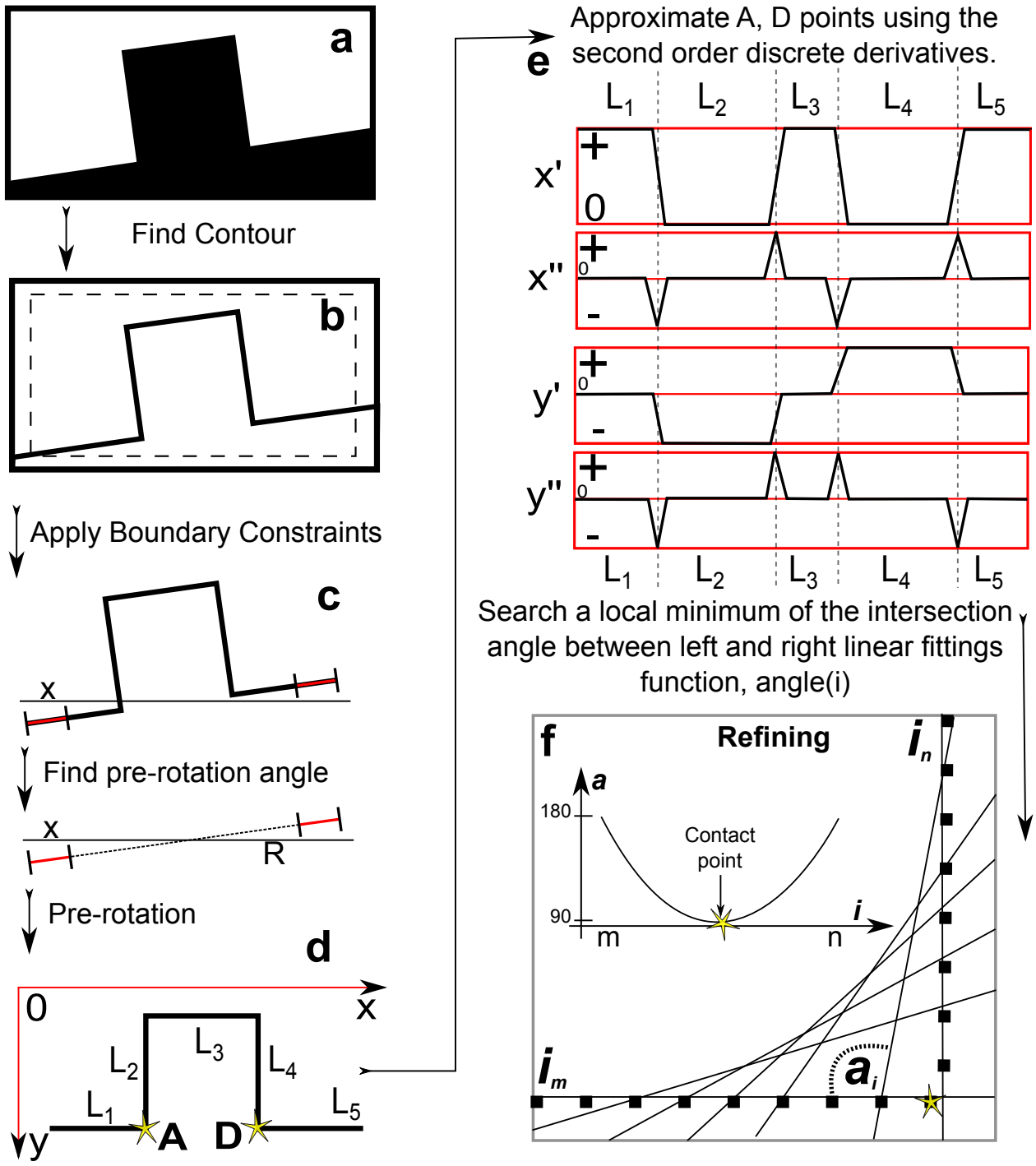
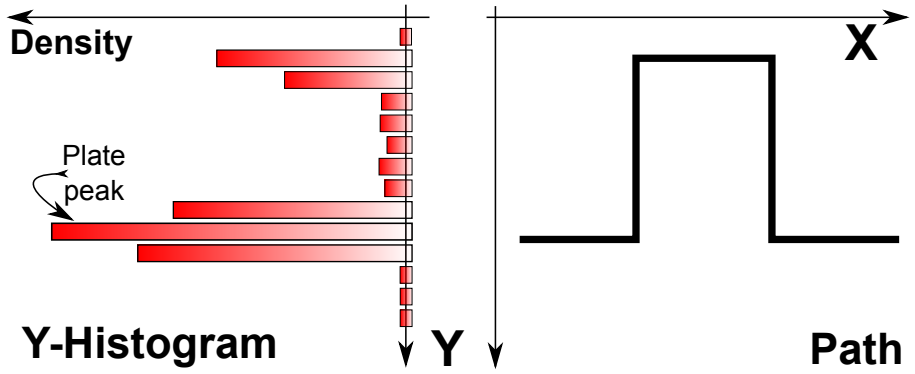


Figure 5.8: Y-Histogram for a simple path



$$
\begin{aligned}
M' &= \begin{bmatrix} x_1 - c_x & x_2 - c_x & \dots & x_n - c_x \\ y_1 - c_y & y_2 - c_y & \dots & y_n - c_y \end{bmatrix} \\
M'' &= \begin{bmatrix} x'_1 & x'_2 & \dots & x'_n \\ y'_1 & y'_2 & \dots & y'_n \end{bmatrix} \begin{bmatrix} \cos a & -\sin a \\ \sin a & \cos a \end{bmatrix} \\
M''' &= \begin{bmatrix} x''_1 + c_x & x''_2 + c_x & \dots & x''_n + c_x \\ y''_1 + c_y & y''_2 + c_y & \dots & y''_n + c_y \end{bmatrix}
\end{aligned}
\tag{5.3}
$$

Al the subsequent calculations are done on this rotated path.

5.4.1.2 A,D Contact Points

Contact points between sample and plate are referred as A for the left contact point, and B for the right contact point. All microscopy and drop analysis is based on those two critical points, as they define *where the real material is*. Their determination can be done in two ways.

Histogram Method The simple *Histogram Method*, implemented in `PathSample.findBaseHist()`, builts an histogram of points classified by their y coordinate, as showed in Figure 5.8.

The biggest density region in the histogram is defined as the plate Y coordinate, P_y. Then it is selected the biggest continuous region of path where all points are above P_y by a minimum threshold e, $y > P_y + e$.

Histogram method is highly robust but someway inaccurate, as the minimum threshold used to identify the sample may erroneously let the algorithm find points which are above the real contact between sample and plate.

Refining may be applied to histogram output in order to search for a better contact point (see below).

Second Order Discrete Derivatives Method A more complex and precise method is the *Second Order Discrete Derivatives Method*, which heavily relies on discrete path

derivatives as defined in 5.2.2. Figure 5.7:e plots up to the second order of path derivatives x', x'', y', y'' for the ideal path in 5.7:d subfigure.

First order derivatives are uninformative: while drawing the plate and the upper side of the sample, the x coordinate increases almost linearly with path index, so x' has a positive plateau. While drawing the sample sides the x coordinate stops changing, so it has a nearly zero plateau. Linear transitions between positive and zero plateau are due to the nature of the discrete derivative function (Equation 5.2).

The y' derivative keeps a zero plateau while the path is drawing the plate, drops to a negative plateau while drawing sample left side, rise to zero again while drawing the upper side of the sample, then rise up to positive values while drawing the right side, and back to zero for the remaining plate segment. Y coordinate sign is due to the inverted Y-axis used as standard for computer imaging.

The second order derivatives instead allow the exact determination of A, D points in most of the cases. The abrupt drops or rises to stable plateaus seen in first order derivatives becomes clear negative and positive peaks.

The A point correspond to a minimum peak over both derivatives, $x'' < 0$, $y'' < 0$, while D point is found at a maximum x'' peak and a minimum y'' peak. More simply, A is the minimum of a $f_A = x'' - y''$ function, while D is the minimum over a $f_D = y'' - x''$ function.

Path imperfections will also create peaks and minimums on second order discrete derivatives, which, for big objects or molten samples, may also be comparable to values found at A,D points. The biggest thing on scene assumptions helps in this case as seen in the Histogram Method: the maximum distance peaks are selected as A,D points.

Another source of errors is the presence of protuberances or holes in the sample, which should never exceed the entity of the sample-plate angle.

Refining After the histogram or derivative contact point search, resulting *raw* A,D points may be refined using a local angle minimizer algorithm, implemented in the `PathSample.findAngle()` method.

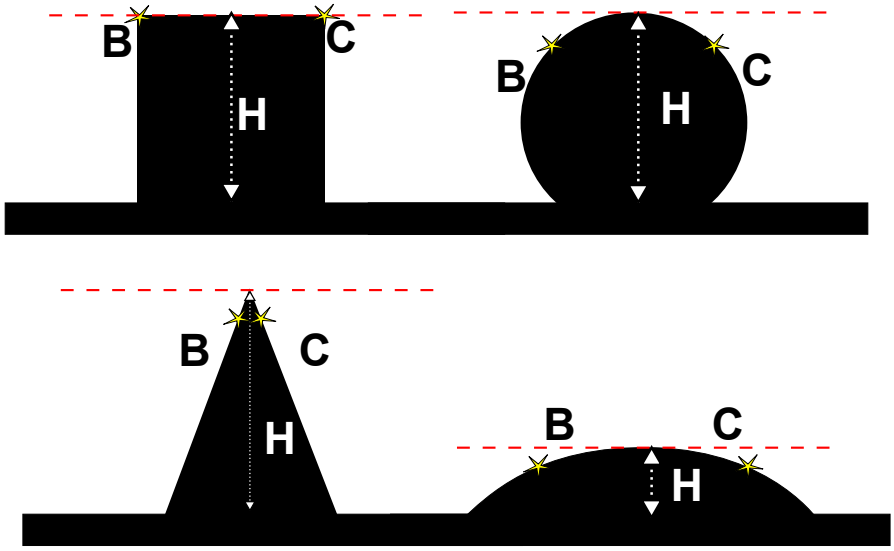
The angle at index i of a path is defined as the angle between two lines, obtained by doing two linear regressions: over n points before i, and over n points after i, as showed in figure 5.7:d. Two lines passes through each squared point, between which an angle is calculated.

A constrained local scalar minimizer [Brent, 1973] searches for the index i showing the smallest absolute angle inside a portion of path around the input i starting index. The algorithm actually finds the maximum convexity point around raw A, D points and returns it as the new, refined contact point.

5.4.2 Advanced Geometrical Properties

A variety of useful geometrical properties can be calculated starting from the detected, filtered, pre-rotated path of the sample material, contained between A, D indexes.

Figure 5.9: B, C upper points determination



5.4.2.1 Dimensional properties

Once the contact points A,D are correctly identified, the algorithm knows which part of the shape pertains to the material and which to the plate.

The **width** (w) property is easily calculated from the distance between the mean mA, mD coordinates, obtained by averaging 6 points around each contact point. It defines the side of the sample touching the plate. The median plate point is calculated from mA, mD coordinates, $x_p = \frac{x_{mA}+x_{mD}}{2}$, $y_p = \frac{y_{mA}+y_{mD}}{2}$. The y_p value is then used to determine the maximum **height** (h) of the sample, by subtracting the minimum y value to it.

Two more critical points are searched, B,C, marking the *upper side* of the sample: as showed in Figure 5.9, an *upper side* can really be defined only for rectangular shapes. In `PathBorder.findPointsBC()`, the minimum of the function $f_B = x + y$ is identified as B, while the maximum of $f_C = x - y$ as C. In other words B is always the uppermost and leftmost point of the path, while C is the uppermost and rightmost point. Thus they are defined for any kind of shape, being it a cone, a sphere, a half-sphere or anything else.

Angles at A,B,C,D critical points are calculated: lower α_A, α_D are the **contact angles** between sample and plate, containing important information about the surface tension of the material and the holding plate. **Upper** α_B, α_C **angles** are used for shape characterization following some standards (see 5.4.3).

Shape **perimeter** L is obtained by summing the distances of each path point from the previous:

$$L = \sum_{k=i_A+1}^{i_D} \sqrt{(x_k - x_{k-1})^2 + (y_k + y_{k-1})^2} \tag{5.4}$$

Area G calculation is demanded to an OpenCV function, `cv.ContourArea`, where the Gauss-Green theorem is applied:

$$G(i_A, i_D) = \frac{1}{2} \int_{i_A}^{i_D} x_i y_i' - y_i x_i' di \tag{5.5}$$

5.4.2.2 Similarity properties

Once the path dimensions are completely determined, more properties helps recognizing its similarity towards reference shapes.

Roundness The roundness ψ property is a percentage index of how the path resembles to a circle of the same area:

$$\psi = \frac{2\pi\sqrt{\frac{G}{\pi}}}{L} \tag{5.6}$$

Center of Mass, Moments and Central Moments First and second order moments $m_{p,q}$ are calculated. Moments are gross characteristics of the path, obtained by the following integration over all points:

$$m_{p,q} = \sum_{k=i_A}^{i_D} x^p y^q \tag{5.7}$$

In moments terminology "order" actually means the to which x and y coordinates are elevated to. For example, the median point coordinates of the path, or **center of mass** (c_x, c_y), are defined as:

$$c_x = \frac{m_{1,0}}{m_{0,0}} \ , \ c_y = \frac{m_{0,1}}{m_{0,0}} \tag{5.8}$$

When moments are calculated after having subtracted path's center of mass, they are called central moments $\mu_{p,q}$:

$$\mu_{p,q} = \sum_{k=i_A}^{i_D} (x - \overline{x})^p (y - \overline{y})^q \tag{5.9}$$

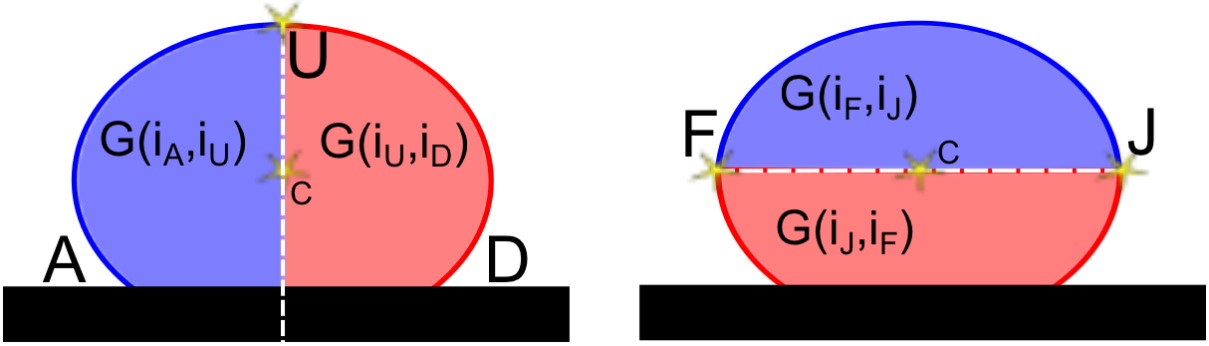
Moments calculation is also demanded to the OpenCV function `cv.Moments`.

Eccentricity The eccentricity of an ellipse is defined as the ratio of the distance between the foci and its major axis length, and is comprised between zero (a circle) and one (a line segment).

Eccentricity of a path can be defined as the eccentricity of the ellipse representing the unit-standard-deviation contour of its points [Zhang et al., 2004], which is computed from the covariance matrix of path points . The covariance matrix Σ is built using path's central moments (eq. 5.9):

$$\Sigma = cov(x, y) = \begin{bmatrix} \mu_{2,0} & \mu_{1,1} \\ \mu_{1,1} & \mu_{0,2} \end{bmatrix} \tag{5.10}$$

Figure 5.10: Vertical and horizontal symmetry properties



The ratio between the two eigenvalues λ_1, λ_2 of the covariance matrix is the eccentricity ε of the path:

$$\begin{bmatrix} \mu_{2,0} & \mu_{1,1} \\ \mu_{1,1} & \mu_{0,2} \end{bmatrix} \mathbf{v} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} \mathbf{v} \qquad (5.11)$$

$$\varepsilon = \frac{\lambda_1}{\lambda_2}$$

Eccentricity will be near zero for compact shapes resembling a circle, and decrease as shape elongation increases.

Vertical and Horizontal Symmetry Vertical and horizontal symmetry properties V_S, H_S expresses how the shape area is distributed towards, respectively, its vertical and horizontal central axes of symmetry, as explained in Figure 5.10.

The vertical ax of symmetry passes through the centroid c_x coordinate and intercepts the upper shape in the_U_point, at index i_U. The area of left portion of contour, going from A to U, is calculated and divided by the area of the right portion of contour, going from U to D:

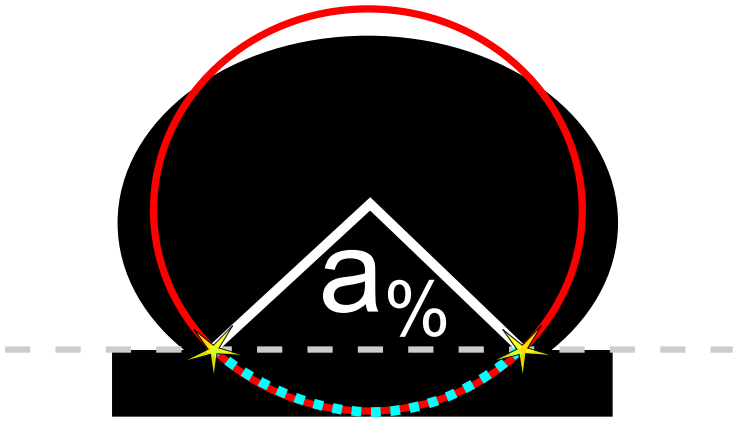
$$V_S = 100 \frac{G(i_A, i_U)}{G(i_U, i_D)} \qquad (5.12)$$

The horizontal axis passes through the c_y centroid coordinate, defining two interceptions with the path: left point F, at index i_F, and right point J, at index i_J. Upper path portion, going from F to J passing through U, is divided by lower path portion, going from F to J passing through A and D:

$$H_S = 100 \frac{G(i_F, i_J)}{G(i_J, i_F)} \qquad (5.13)$$

Circle Fitting The `PathSample.calcCircleFit()` method actually tries to determine the best circle fitting the path. It initially estimates the centroid (eq.5.8) and the radius by meaning the distance of every point from the centroid (eq. 5.14).

Figure 5.11: Angle defining the non-existing fitted circle part



$$r_0 = \frac{\sum_{k=i_A}^{i_D} \sqrt{(x_k - c_x)^2 + (y_k - c_y)^2}}{L} \tag{5.14}$$

The number of points in the path is reduced according to a configurable `precision` option, to reduce fitting duration. The fitting itself is performed with a Least Squares Regression method of the SciPy library. It returns the **fitted center** Θ and **radius** r values, and a **fitting error** e_Θ.

The percentage of theoretical circle really existing is then calculated by cutting the circle with the base, then calculating the angle $\alpha_{\%}$ between the intersecting points and the center, as showed in Figure 5.11. That angle is then proportioned to 2π in order to express the **existing circle part** as a percentage $c_{\%}$:

$$c_{\%} = 100(1 - \frac{2\pi - \alpha_{\%}}{2\pi}) \tag{5.15}$$

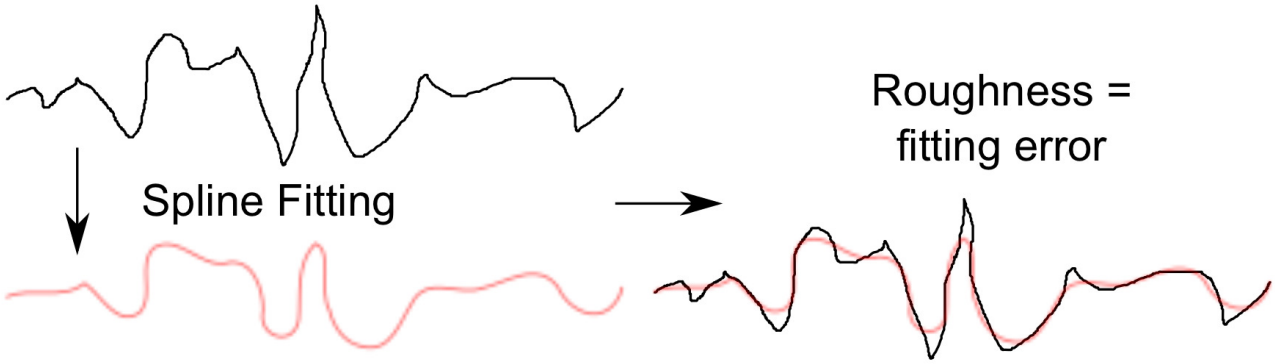
Roughness Path roughness is an important property for determining how material's surface tension and viscosity collaborate in canceling small imperfections, like holes, protuberances, edges. An index of how much a path is rough is calculated by fitting it with a *spline* having a lower number of nodes than path's points. Points/nodes ratio and the order of spline polynomials can be configured `precision` and `grade` options.

Cohesion Cohesion is probably the most meaningful path property. It is expressed as a function of the relative standard deviation of points distances from the center of mass (or centroid) of the path. Let vector d contain distances between each point of the path and path's centroid:

$$d_i = \sqrt{(x_i - c_x)^2 + (y_i - c_y)^2} \tag{5.16}$$

The cohesion property Ω is defined through the ratio between d standard deviation, σ_d, and d mean value $\overline{d}$, that is, the *relative standard deviation* of d. This

Figure 5.12: Roughness expressed as spline-fitting error



ratio is expressed as a percentage number where zero error gets $\Omega = 100\%$, and all other values gets lower cohesion:

$$\Omega = 100(1 - \frac{\sigma_d}{\bar{d}}) \tag{5.17}$$

5.4.2.3 Solids of Revolution

If the assumption that the sample has a vertical axis of symmetry is satisfied, three-dimensional properties can be inferred from a two-dimensional shadow path by rotating it around that axis.

A Solid of Revolution is generated by rotating a plane curve around an external axis of revolution. Surface and volume of that solid can then be calculated according to, respectively, First and Second Pappus's Theorems.

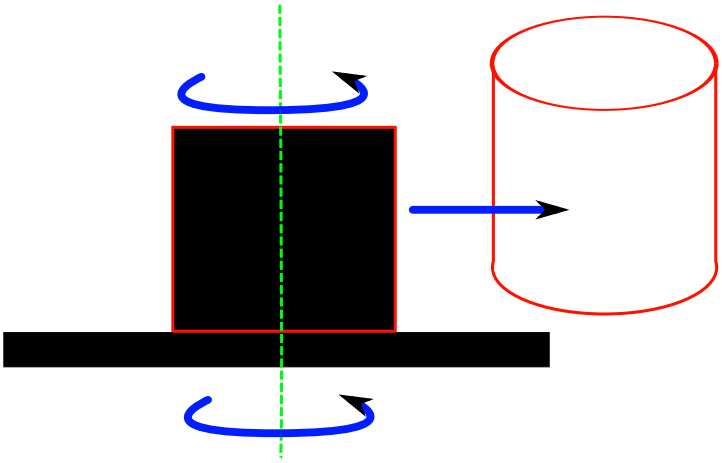
- The **surface** S of a solid of revolution is the arc length of the curve multiplied by the distance traveled by its geometric centroid during the revolution.
- The **volume** V of a solid of revolution is the area of the curve multiplied by the distance traveled by its geometric centroid during the revolution.

The problem is that our revolution axis is not outside of the path, but, in order to reconstruct the three-dimensional sample, must pass exactly through path's centroid, as showed in Figure 5.13. Thus, the path is divided in two portions, left and right to the axis. For each portion the area and center of mass are found, then the side is rotated and the two resulting partial volumes and surfaces are summed up to give total S and V.

Sphericity A sphericity property can be calculated from volume and surface. Sphericity is the three-dimensional translation of the bi-dimensional roundness: the ration between the surface of a sphere of equivalent volume over the real surface:

$$\Psi = \frac{\pi^{\frac{1}{3}}(6V)^{\frac{2}{3}}}{S} \tag{5.18}$$

Figure 5.13: Solid of revolution around a vertical axis passing through the centroid



5.4.3 Heating Microscopy Shape Characterization

Misura image analysis method introduces many properties which are not considered in current international standards (Table 5.2). For that reason, their meaning and their association with characteristic shapes had to be established.

A systematic experimental evaluation is postponed to future scientific publications: only the physical considerations leading to the Misura method of determining characteristic temperatures (CT) is herein explained. Geometrical properties are considered one-by-one, and their relation to shape and physical transformations of the sample material is discussed.

5.4.3.1 Indicator Property Definition

A good indicator property should be invariant to the following test execution conditions, which are not related to the material behavior (*external conditions*):

Rotation. Rotation effects are almost completely filtered out by the pre-rotation applied to every analyzed frame (see 5.4.1.1). All presented indicators are rotation-safe.

Resolution. The number of pixel forming the shape. Zooming or different camera sensors produces images composed of a varying number of pixels, where details are more or less defined. It is assumed the algorithm knows the real length corresponding to one pixel (*pixel calibration*). Perimeter, for example, does depend on resolution as more surface defects are visible at higher resolutions. On the contrary, height is invariant as it is normalized by pixel calibration.

Initial Geometry. Each standard specify its own initial shapes, moreover the operator may need to analyze custom samples. The same CT should always be recognized, so initial geometry should have the minimal influence on the behavior of the indicator property.

Table 5.3: External conditions dependence of properties

Property	Symbol	Local	Init. Geom.	Prep.	Res.
Contact Angles	α_A, α_D	A,D	Should not	Yes	-
Upper Angles	α_B, α_C	B,C	Yes	Yes	-
Width	w	A,D	Yes	-	-
Height	h	A,D	Yes	Yes	-
Perimeter	L	A,D	Yes	Yes	Yes
Area	G	A,D	Yes	-	-
Roundness	ψ	-	-	Yes	Yes
Surface	S	A,D	Yes	Yes	Yes
Volume	V	A,D	Yes	-	-
Sphericity	Ψ	-	-	Yes	Yes
Eccentricity	ε	-	Yes	-	-
Horizontal Sym.	H_S	-	Yes	-	-
Fitted Radius	r	-	-	-	-
Existing Circle	$c_{\%}$	A,D	Yes	-	-
Fitting Error	e_Θ	-	Yes	-	-
Roughness	rgn	-	-	Yes	Yes
Cohesion	Ω	-	-	-	-

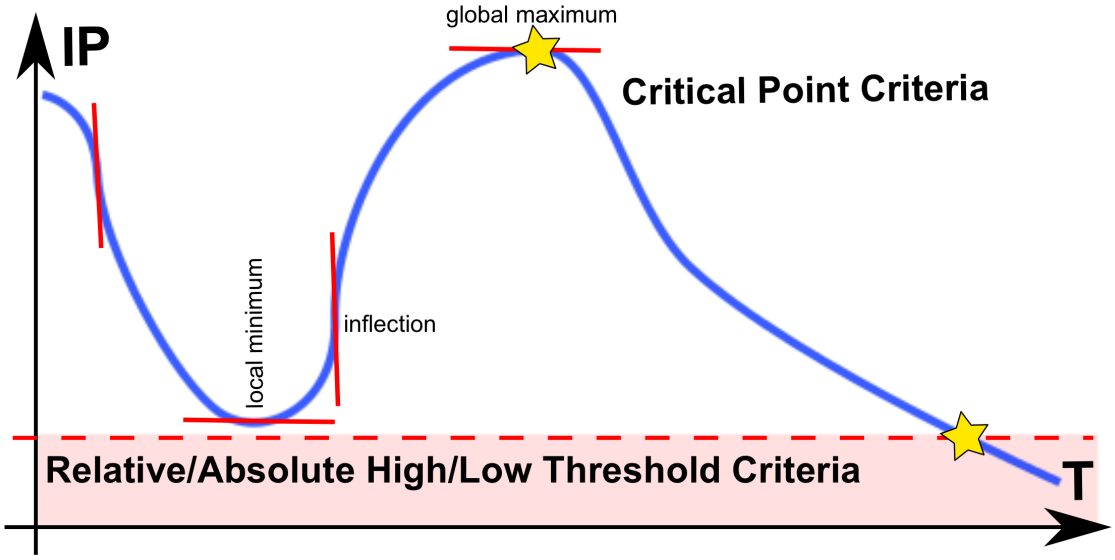
Obviously, absolute values of all properties *must* depend on starting conditions. Otherwise the indicator would not depend on subsequent critical geometries neither and be totally useless for recognition. But the indicator's *behavior*, intended derivative signs in certain shape transitions and critical points, should be *initial-geometry-invariant*.

Sample Preparation. Preparation is not an easily reproducible operation, and may lead to very different samples shapes. Borders, and their angles, are the less reliable geometrical features. Granularity and cohesiveness differences amongst materials, together with variable wetting and pressing applied to powders, determine dramatic differences in the starting surface rugosity.

Local effects. The indicator cannot rely on a single or few points of the path, but should be the result of a calculation involving the whole path. Failing this requisite will most likely lead to unreproducible characterizations. For example, contact angles are not good fusion indicators, as they rely on few tens of points of the shape. Also any other property heavily relying on A, D identification may experience instability. Indeed, if something weird happens near these few points, the characterization fails, where an operator would not even need a second look to recognize the shape.

Table 5.3 details dependence from external conditions of all properties defined in Misura 4.

Figure 5.14: Criteria being applied to an indicator property for shape recognition



5.4.3.2 Numerical Criteria for Shape Recognition

An indicator property should change in a predictable way during the test. Characteristic shapes are recognized by setting numerical criteria on the IP, as showed in Figure 5.14.

Thresholds When a property value goes over or under a threshold, a characteristic temperature is recognized. Threshold themselves may derive from other property values (relative) or be absolute. For example, all international standards apply a minimum relative threshold criteria to the height property for fusion determination. The threshold itself should be calculated from height and width values of the precedent characteristic shape (the half-sphere).

The CEN standards specify an absolute threshold for the roundness to be achieved for softening determination. When the roundness increases by more than 15% of the initial value, softening is identified.

Critical Points The characteristic shape is recognized whenever the property reaches its global maximum or minimum value. They are not used in current international standards.

5.4.3.3 Indicator Properties and Characterization

External conditions *invariance* (see 5.4.3.1) is a desirable feature of a proper indicator, but the most important one is its *variance* with shape transformations, summarized in Table 5.4. Cell colors refers to important changes in property derivative, which probably lead to global maxima (cyan followed by magenta) or global minima (magenta followed by cyan).

Table 5.4: Behavior of properties approaching characteristic shapes

Property	Symbol	Sinter.	Soften.	Sphere	H-Sphere	Fusion
Contact Angles	α_A, α_D	-	-	⇓	⇑	⇑
Upper Angles	α_B, α_C	-	⇓	⇓	⇓	⇓⇓
Width	w	-	-	-	-	⇑⇑
Height	h	⇓	~	~	⇓	⇓⇓
Perimeter	L	⇓	⇓⇓	⇓	~	~
Area	G	⇓	-	⇓	⇓	⇓⇓
Roundness	ψ	-	⇑	⇑	⇓	⇓⇓
Surface	S	⇓	⇓⇓	⇓	~	~
Volume	V	⇓	-	-	-	-
Sphericity	Ψ	-	⇑	⇑⇑	⇓	⇓⇓
Eccentricity	ε	-	~	⇓	⇑	⇑⇑
Horizontal Sym.	H_S	-	⇓	⇑	⇓	⇓⇓
Fitted Radius	r	⇓	-	⇓	⇑	⇑⇑
Existing Circle	$c_{\%}$	-	⇑	⇑	⇓	⇓⇓
Fitting Error	e_Θ	-	⇓	⇓	⇓⇓	⇓⇓
Roughness	rgn	-	⇓	⇓	-	-
Cohesion	Ω	-	⇑	⇑⇑	⇓	⇓⇓

Sinterization or Shrinkage Sinterization or shrinkage of a powder is an isotropic transformation, where the shape of the object is maintained while its dimensions diminish proportionally in each considered direction in space.

For anisotropic (and monolithic) samples, height/width ratio may slightly change, but fundamental shape features are maintained.

Altough CEN standard impose the **area** indicator, it must be noted as area - and area change - highly depends on the initial geometry of the sample. The percentile change in area $\delta G_{\%}$ of a cone subject to shrinkage would be substantially different from that of a cylinder, a trunk cylinder or a trunk cone. It would also be different if the height/width proportion of that initial geometry vary amongst measurements.

If the vertical symmetry axis assumption is true, **Volume** is a much proper sinterization indicator. Volume extrapolation automatically takes care of the effect a shadow change has, by considering also the distance it have from the rotation axis.

Thus, a percentile volume change criteria $\delta V_{\%}$ is applied in order to establish if shrinkage has been achieved (usually, 5%).

Deformation or Softening Standard Deformation or Softening identification is demanded to operator's experience, as "first rounding" can hardly be considered a mathematical criterion. Only ASTM and CEN norms try to formalize with numbers this characteristic shape.

ASTM checks whenever the height becomes equal to width: it depends on initial geometry *by definition,* so cannot be considered satisfying for Misura's generalization ambitions.

CEN Standards use the roundness indicator property, checking if it increases by

more than 15%. While this is a much more general requirement than ASTM criterion, it is still not enough independent from initial shape.

The softening indicator should be normalized not only towards the starting shape of the sample, but also towards the *ending* shape of the softening phase: the sphere.

Compare the roundness of a $h = w$ cylinder C1 and a $h = 2w$ C2 cone:

Shape	Sym.	Proportions	Roundness	Target
Cylinder	C1	h=w	83.5 %	96.0 %
Cone	C2	h=2w	64.8 %	74.5 %

C1 starts from a very high roundness, so a 15% increase means a big difference. C2 starts with a low roundness value, so 15% is a smaller absolute increase.

The physical behavior of the sample further aggravate this dependence from initial geometry. Gravity, combined with lowering viscosity, will tend to press and expand the sample on the plate (flowing). If the starting shape is vertically elongated, it will have a bigger effect and make roundness increase quicker than if the shape is horizontally elongated or not elongated at all.

Moreover, if the material starts in a small-surface disposition, surface tension will emerge with less evidence.

Misura not only evaluates not only the starting shape of the sample, but also the spherical shape, that is, the maximum roundness at which a shape can be deformed by material's surface tension and by gravity. Roundness to be 2/3 of the way to the spherical value:

$$\psi_D = \psi_0 + \frac{2}{3}(\psi_\Theta - \psi_0) \tag{5.19}$$

Sphericity or Cohesion are also proper indicators:

$$\Psi_D = \Psi_0 + \frac{2}{3}(\Psi_S - \Psi_0) \tag{5.20}$$

$$\Omega_D = \Omega_0 + \frac{2}{3}(\Omega_S - \Omega_0) \tag{5.21}$$

In order to correctly identify softening it should also be known not only the starting roundness of a sample, but also which is it's maximum value.

A sinterization cost function is defined, which considers the proximity of Roundness, Sphericity and Cohesion values to their target sinterization value:

$$E_S = (\psi - \psi_D)^2 \cdot w_\psi^D + (\Psi - \Psi_D)^2 \cdot w_\Psi^D + (\Omega - \Omega_D)^2 \cdot w_\Omega^D \tag{5.22}$$

Proper weight factor (w_{var}^D) may be customized following user needs.

Sphere A critical point criterion may be formulated for spherical characteristic temperature identification.

As evidenced by Table 5.4, this CT induces a sign change in many properties, and for some of them it corresponds to a critical point.

Contact angles are a local property calculated from few points (ref. Table 5.3), thus should be discarded as not very reliable.

Roundness, Sphericity, Eccentricity, Radius, Existing Circle and Cohesion are all possible targets for a critical point criterion.

Misura 4 defines a spherical cost function by summing all sphere-related properties at once, multiplied by proper weighting factors (w^S_{var}):

$$E_S = -\psi \cdot w^S_\psi - \Psi \cdot w^S_\Psi + \varepsilon \cdot w^S_\varepsilon + r \cdot w^S_r - \Omega \cdot w^S_\Omega \tag{5.23}$$

This cost function is then minimized in order to find the best spherical point, according to all these possible and valuable indicators.

Half Sphere The half-sphere is easily determined by the Existing Circle property, which, according to its definition, reaches 50%:
$c_\% = 50\%$

Fusion Fusion is the characteristic shape of easiest recognition, because of the marked geometrical change showed by the liquid drop flowing on the holding plate. Almost any property may be chosen to identify this value. The standards rely on the simplest one, height, requiring it to go under a relative threshold.

Two properties with opposite fusion behavior can also be combined, for example radius or width and height, and further enhance their change by considering their ratio. While height drops near the level of the plate, width extends to cover the maximum extent of the plate: a very robust relative threshold may then be applied to $\frac{h}{w}$.

5.4.3.4 The All-In-One Indicator: Cohesion

The Roundness and Sphericity apparently tell how much an object is similar to a sphere. Thus, both are used in Deformation and Sphere determination.

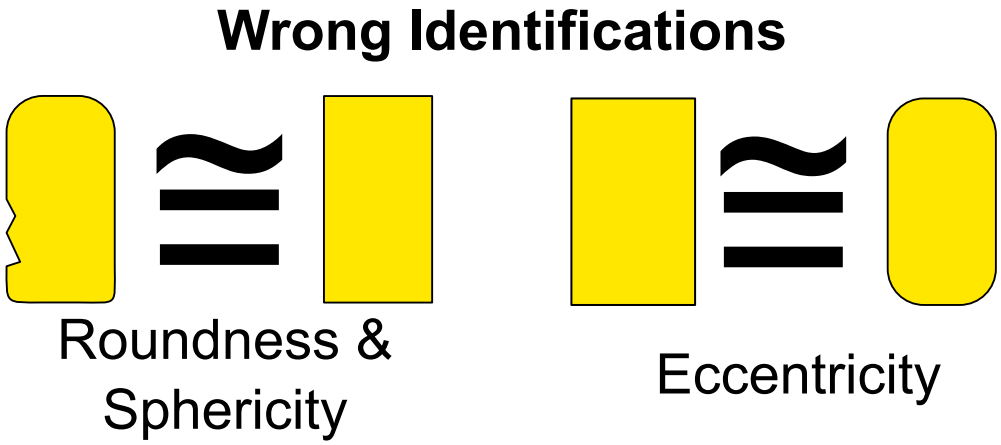
Anyway, if assumption 6 is accepted - object has a vertical axis of symmetry - sphericity would be a more realistic indicator than roundness, as explained in Sinterization description. It will take into account that a border modification in a region which is far from the axis has a much greater effect than a change which happens near the axis.

But sphericity also highly depends on border roughness, as defects are amplified by rotation and integration, and are also weighted by their axis distance. So it may be used as a sphere indicator *only after the deformation point*, where all surface defects should disappear - and is not a good Deformation indicator at all.

Both roundness and sphericity directly or indirectly rely on the perimeter/area ratio. The strong perimeter dependence on border roughness introduces another defect (Figure 5.15, left). Neither perimeter nor area distinguish if their value is due to surface defects, or to shape features, like macroscopic edges.

For example, a perfect rectangular profile may have the same roundness and sphericity compared to an imperfect circle with some microscopic surface defects.

Figure 5.15: Common Roundness, Sphericity and Eccentricity identification errors



Eccentricity gives a better indication of spheric shape, but totally disregards border defects and shape's features (Figure 5.15, right).

First signs of rounding of a rectangle's edges will have a negligible effect over eccentricity. So, while eccentricity is a good and robust sphere indicator, it will be of little or no use in Deformation identification.

Cohesion was purposely introduced to overcome those limitations. It represents a relative standard deviation of points distances from the center of mass (eq. 5.17) , so:

- It has a lower dependence on local surface defects than roundness or sphericity.
- Will show significant variations as macroscopic shape features change, like an entire edge disappearing - unlike eccentricity.
- Sintering does not have any effect at all. So the end of shrinkage can be identified by checking when cohesion starts to vary.

Cost functions presented in Deformation and Sphere may actually consider this indicator alone.

Cohesion is the most clear and robust indicator for Deformation, Sphere and even Fusion and Sintering identification.

Chapter 6

Thermal Control

Behavior and properties of materials are commonly highly dependent on the thermal treatments they undergone. The ability to flexibly control the heating cycle of the sample under study is a key feature in any thermal analysis instrument.

The market offers devices which can automatically control the temperature and heating rate of a furnace, once properly parametrized. Misura 3 software relied on an external PLC, managing up to 16 linear *ramps* of temperature. The electrical power output to the furnace is automatically controlled by an industry-standard Proportional-Integral-Derivative (PID) algorithm, matching the required temperature curve fairly well.

Advances in material science now require the operator to realize complex heating cycles, composed of many of such linear ramps or comprising non-linear temperature curves. New investigations would be possible if the temperature curve could be calculated based on measured material's properties. An example application is the realization of rate-controlled sintering, applied in the production of many high technology ceramics and materials [Palmour et al., 1974, Ragulya and Skorokhod, 1995, Greil, 1989, Ragulya and Skorokhod, 1998, Kaisersberger et al., 1988, Speyer et al., 1992].

Such advanced and highly specific behaviors cannot be obtained with a generic-purpose industrial device.

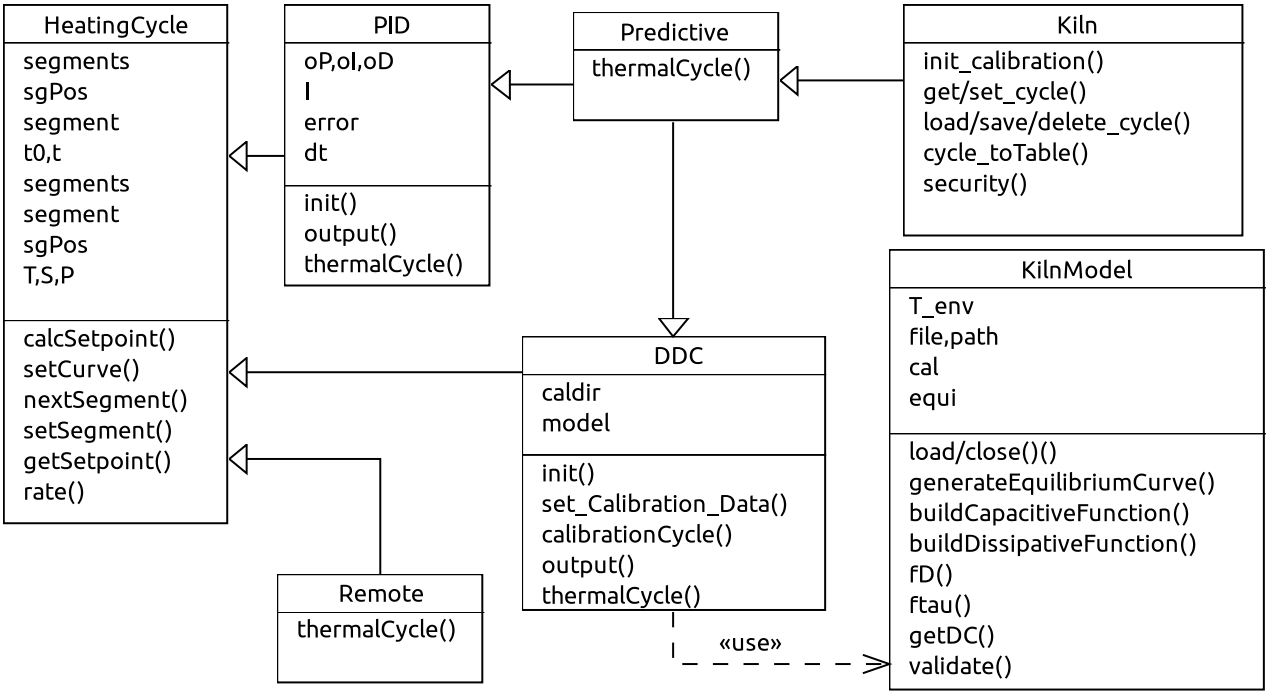
Misura 4 implements its own P.I.D algorithm. The PID concept is further extended by formulating a physical model of the furnace, capable of predicting and approximating the power needed to obtain any temperature curve - not limited to ramps.

6.1 Implementation

Misura implements *the Kiln* as a standalone instrument, controlling all thermocouples and the power unit. `Kiln` performs its very unique measurements, consisting in controlled heating cycles, by reading temperature inputs and controlling the power output in order to obtain the desired heating cycle.

The classes implementing this functionality are modeled in Figure 6.1.

Figure 6.1: Thermal control Class Diagram



6.1.1 The Kiln Instrument

The `Kiln` class inherits from both the general `Instrument` class and from the `Predictive` regulator class.

It defines an option, `cycle`, and two set/get layer methods. The Client refers to this option in order to get the current thermal cycle or to change it to a new one. Method `save_cycle` will save the current cycle to a file, while `load_cycle` reads it back and `delete_cycle` deletes the file from the disk. This enables the user to save and reload by filename commonly used thermal cycles.

The `cycle_toTable` method is called during the `Instrument.init_storage` procedure, in order to save the cycle into the test file.

The `security` method will check current and recent temperature, power and setpoint, and block the heating if something dangerous happened: for example, if the temperature is too far from the setpoint, or if its derivative is too high.

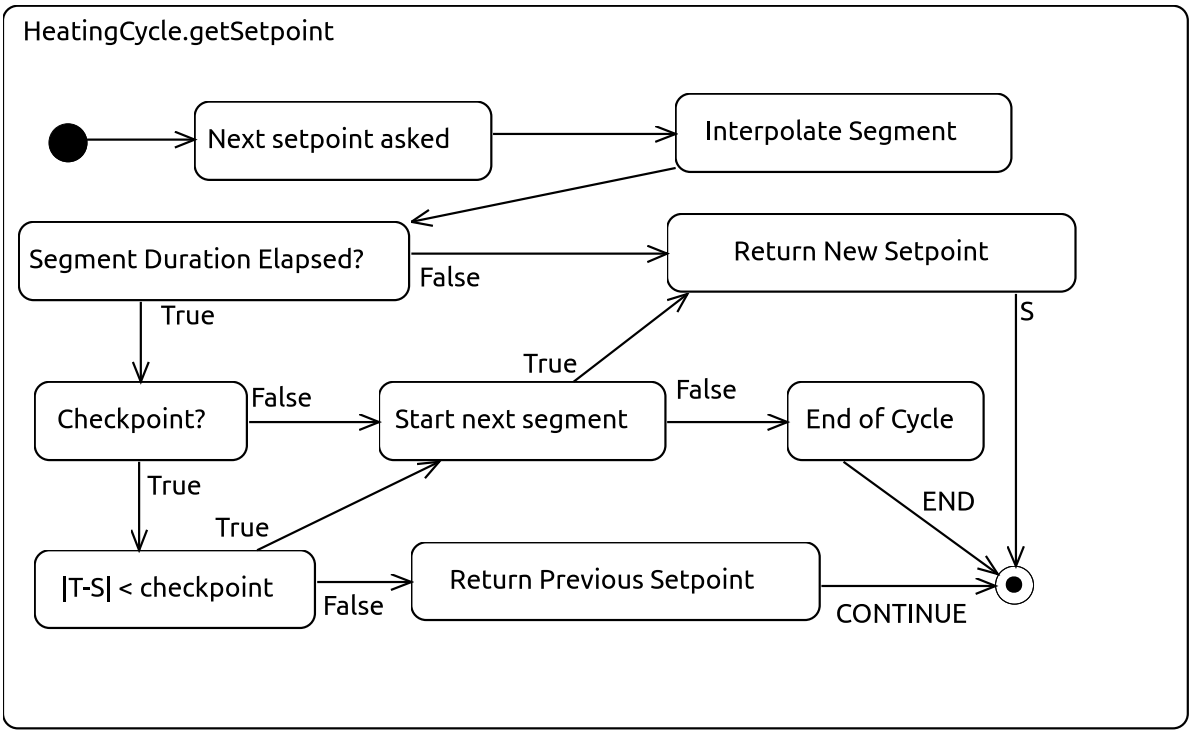
The rest of the `Kiln` functionality is inherited from parent classes.

`Kiln` is a very special object, because it can serve as a standalone `Instrument`, but *is used* by other instrument as a `Device`. So it both implements a `supervisor_iteration` method, called during `Instrument`-like operation, and a `run_acquisition` method, called during `Device`-like operation.

6.1.2 HeatingCycle Ramps Management

`HeatingCycle` is a pure Python `object` child class, and does not inherits any other Misura class. Its sole target is setpoint management, which means to correctly determine the desired temperature as a function of time and the configured `curve`.

Figure 6.2: Setpoint Activity Diagram



Setpoint generation is modeled in the Activity Diagram 6.2.

During temperature control, regulator classes ask which is the current setpoint and heating rate to be reached, in order to calculate a proper output power value to be transmitted to the power unit.

6.1.2.1 Segments and Checkpoints

The first task of the `HeatingCycle` object is to read the desired heating treatment, validate it and split it into *segments*.

An heat treatment is defined by a sequence of (t, S, chk) points, where t stands by time, S by setpoint and `chk` by checkpoint. The (t, S) curve is the theoretical, target heating cycle to be realized.

The real curve will certainly differ, due to inherent nature of any automatic process control The checkpoint chk value asks to the process control to wait until $|S - T| < chk$. When the error between setpoint and temperature falls under the chk tolerance, `HeatingCycle` can proceed executing the next segment.

This feature is very useful if the operator needs to absolutely reach a maximum target temperature and then control the cooling. Without the checkpoint option the kiln will never exactly reach the target, as exasperate in Figure 6.3.

Segments are defined by checkpoints. An arbitrarily complex heating cycle which does not contain any checkpoint is considered a single segment.

The left curve S_1 in Figure 6.3 and Table does not contain any checkpoint, and it is represented as a single `HeatingCycle` segment. The right curve, instead, contain a checkpoint, so it is represented by two segments (cyan and magenta).

Figure 6.3: Checkpoint feature of heating cycles

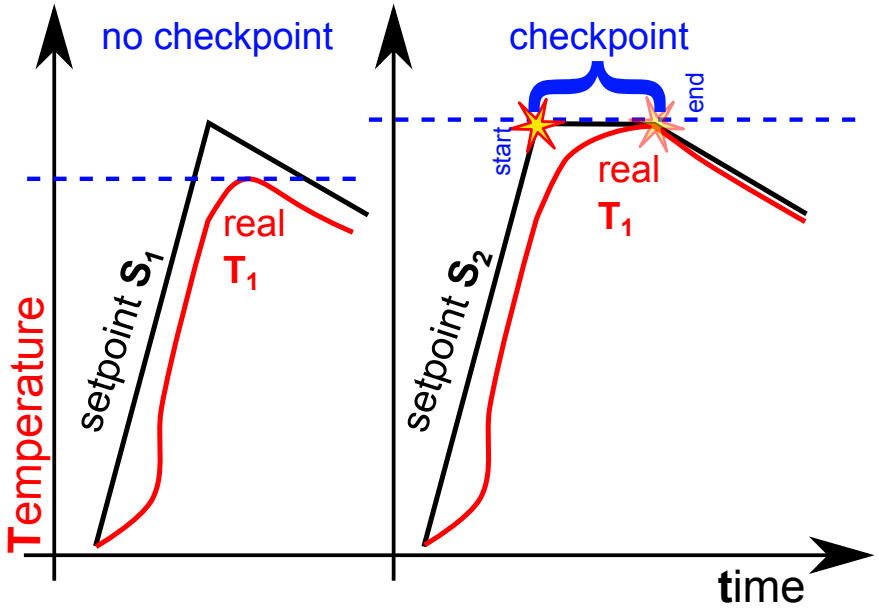


Table 6.1: Heating cycle curves for Figure 6.3.

t (min)	S_1 (°C)	chk_1(Δ°C)
0	0	0
30	1000	0
70	800	0

t (min)	S_2 (°C)	chk_2 (Δ°C)
0	0	0
30	1000	**3**
70	800	0

6.1.2.2 Input Curve Validation

Validation helps avoiding against erroneous input curves. The most important correction is to reduce too high heating rate parts. A maximum heating rate is configured for each kind of kiln, which should never be surpassed. Upon user curve configuration, high heating rate curve parts are slowed down to the maximum admissible rate, as showed in Figure 6.4.

6.1.2.3 Workflow and Interpolating Functions

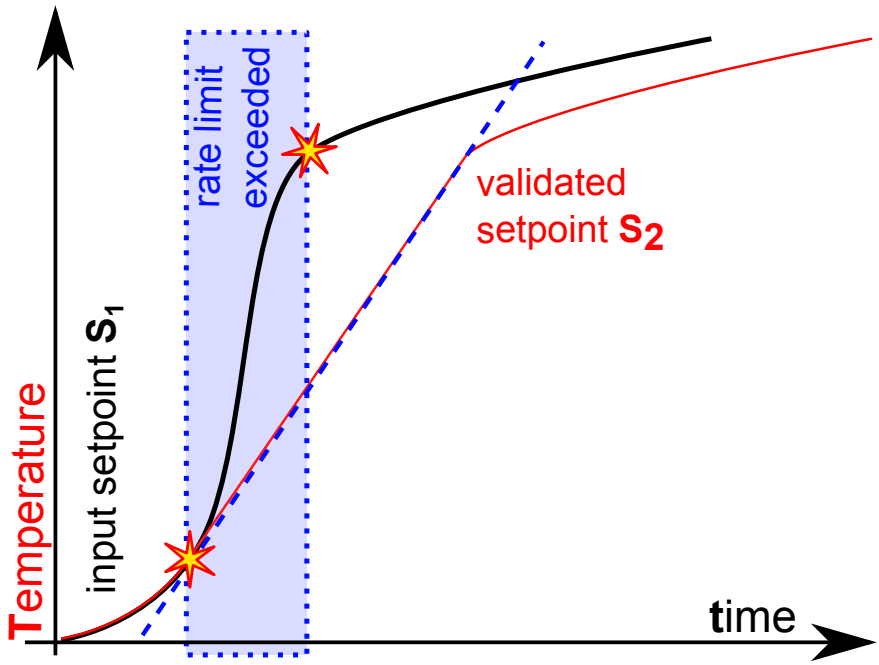
When a regulator class asks for the current setpoint and rate (`getSetpoint` method), `HeatingCycle` will get the segment which is currently executed. If the segment time duration is elapsed and there is not a checkpoint, then the segment is marked as ended and the next segment is searched for (`nextSegment` method). If no new segments are found, the thermal cycle ends and `HeatingCycle` returns an `END` message.

If the next segment exists, then its sequence of $(t, T, 0)$ points is read and a linear interpolating function `calcSetpoint(t)` is created.

This interpolating function accepts a time argument, and returns the current setpoint calculated by linear interpolation between the points defining the segment.

The `calcSetpoint` function is also used to calculate the current heating rate, by differencing current and one-second-ago setpoints.

Figure 6.4: Validation effect over a too quick heating cycle.



When a segment ends and the checkpoint value is $chk > 0$, `HeatingCycle` waits for the temperature-setpoint error to drop below the threshold defined by the chk value itself. A special `CONTINUE` message is returned, meaning that any previous setpoint and rate value should be maintained.

6.1.3 An Interface for Automatic Regulation

The real process control is implemented in four classes, called *regulators*: `Remote`, `PID`, `DDC`, `Predictive`. All of them implement the `thermalCycle` method, which is called by `Kiln` during both its `Device`-like and `Instrument`-like operation.

`Remote` is the simplest regulator class, as it simply route the setpoint request to an external device, like a thermoregulator, which will then control the process by itself.

Next sections will discuss `PID`, `DDC` and `Predictive` more advanced characteristics.

6.2 Traditional P.I.D. Regulator

PID abbreviation stands for the three feedback mechanisms of this regulator: Proportional, Integral and Derivative. PID is the preferred control algorithm for 95% of industrial processes, and finds wide application in a variety of fields requiring automatic process control.

During each `thermalCycle` control iteration, the **error** $e(t)$ between the *process value*, temperature, and the target *setpoint*, the heating cycle, is calculated:

$$e(t) = S(t) - T(t)$$

The output power is calculated by summing three values derived from this error:

the proportional output o_P, the integral output o_I and the derivative output o_D.

$$O = o_P + o_I + o_D$$

6.2.1 Proportional Output Power

The proportional output power o_P at a given time t is calculated by multiplying the error by a constant, the proportional factor K_P:

$$o_P(t) = K_P \cdot e(t) \tag{6.1}$$

As the error increases, the proportional power absolute value increases, while its sign is determined by the sign of the error. If the temperature is below the setpoint, o_P is positive, otherwise it is negative.

6.2.2 Integral Output Power

This output component is proportional to the integral of errors committed in the past, by the integral factor K_I:

$$o_I(t) = K_I \cdot \int_0^t e(t) \cdot dt \tag{6.2}$$

If $o_I = 0$ the controller will probably reach an equilibrium condition of constant error, which allows o_P to be sufficiently high for the temperature to increase at the same rate as the setpoint, but is not enough to totally remove the error.

The integral components corrects this error by considering is duration in time.

6.2.3 Derivative Output Power

While integral component sums up past errors, the derivative component tries to forecast future errors and apply the proper output power feedback.

The o_D is proportional to the derivative of the error curve, by the derivative factor K_D:

$$o_D = K_D \cdot \frac{de}{dt} \tag{6.3}$$

Without a derivative term, the controller may be too slow reacting to quick temperature increases or drops. In practical measurement instruments application, derivative reaction has little or no effect, as quick temperature changes are usually unwanted. On the contrary, they are precious signals of malfunctioning, thus invalidating the entire measurement, or of material's phase transitions, which should be noticed by the operator and not automatically corrected.

6.2.4 Defects and corrections

The general applicability of the PID algorithm leads to two main defects: the runaway risk and the temperature delay/oscillation due to thermal inertia.

The runaway implies the power to go to 100%, or quickly drop down and step up, under exceptional conditions. This sudden power increase will cause some stress over any heating element, reducing its duration and performance, or even breaking it for thermal shock.

Runaway is partially avoided thanks to security checks over temperature behavior with respect to setpoint and outputted power. Anyway, it cannot be totally avoided, and 100% output will anyway be applied at least in the first seconds of those exceptional events.

Moreover, a set of power boundaries may be enforced as a function of temperature, in order to avoid especially dangerous high power values at low temperatures.

The temperature delay and oscillation are two complementary effects. Both are caused by two concurrent factors, a physical and a mathematical one:

- **Physical**. Measured temperature will not instantly react to an output power, as the heating element needs few milliseconds to heat up, then the heat has to travel from the element to the sensor, which may require minutes.

- **Mathematical**. At the beginning of a cycle the error is zero, so all PID addenda are null. A consistent error *must be committed* before the controller starts correcting it. Moreover, the integral component of the PID algorithm *needs errors to accumulate* over a time span, so it will act with further delay than o_P, o_D.

Due to these limitations, if the algorithm is configured in order to immediately react to an error, it will lead to oscillations - which would render a measurement instrument unusable. Misura instruments, instead, configure the PID controller in order to accept a constant delay of temperature, but precisely realize the heating rate asked by the user. For this reason the checkpoint concept was introduced in the `HeatingCycle` class (Figure 6.3).

6.2.4.1 Implementation

PID algorithm implementation is quite simple. Control factors K_P, K_I, K_D are exposed to the user as options. Error integral and derivative functions are built accessing chronological T, S values stored in their `CircularBuffers`. Finally, a cutoff limit is imposed to the resulting power, forcing it between 0% and 100%.

6.3 Developing the Dissipative-Capacitive Regulator

While the PID algorithm is generally considered the best control algorithm for unknown system, its defects may be mitigated if the system can be previously studied. If a theoretical model exists for the system, and is well parametrized, then

the output power may be deterministically calculated from the current process value and the setpoint curve.

This kind of control is *open-loop*, and totally disregards the real temperature value. Thus, the runaway PID defect would be totally avoided, as external events are not considered. Also the mathematical part of the initial temperature delay is solved, as the algorithm does not need to commit an error in order to control the power. It doesn't even *know* if an error is being committed, because it does not evaluate the temperature.

The control problem is in reality transformed into a modeling problem. There should exist a good enough model describing the thermal behavior of the heating system.

Unfortunately, this is impossible in practice. The sample material introduces a set of unpredictable variables in the model. For example, the model should know the thermal capacity behavior of the sample, and exothermic or endothermic reactions occurring as a function of time and temperature.If all those variable were already known to the operator, it would probably be pointless any further thermal analysis on that material.

A reasonable model will anyway help correcting some of the PID algorithm defects, if used in combination.

The model formulated for Misura furnaces is based on Fourier's Law for Heat Transfer describing heat conduction through materials, and the Newton's Law of Cooling describing the free cooling of an object in air.

6.3.1 Stationary Calibration

The input power needed to maintain a certain temperature can be easily determined with a calibration procedure, modeled in Activity Diagram 6.2.

The cold furnace receives a constant power input until it reaches a stationary condition. When temperature derivative decreases under a configurable value, a new equilibrium point T_E, P_E is recorded. The input power is then raised by a step (eg, 10%), and a new stationary point is waited for.

When the power or the temperature reach their maximum threshold, usually (70%, 1400°C), the procedure is ended, and the free cooling curve is recorded.

An approximate stationary heating controller may be obtained by interpolating P_E points as a function of equilibrium temperatures, T_E. A stationary power interpolating function $P_E(T)$ is defined, which returns the power required to maintain any desired temperature.

This simple controller only accounts for steady-state heat dissipations. It will not allow to realize any ramp, and the real temperature will take a very long time to reach even a constant setpoint.

In order to realize ramps, we should know how the temperature changes with the input power. This can be derived from the heat capacity C of the furnace:

Table 6.2: Stationary Calibration Activity Diagram

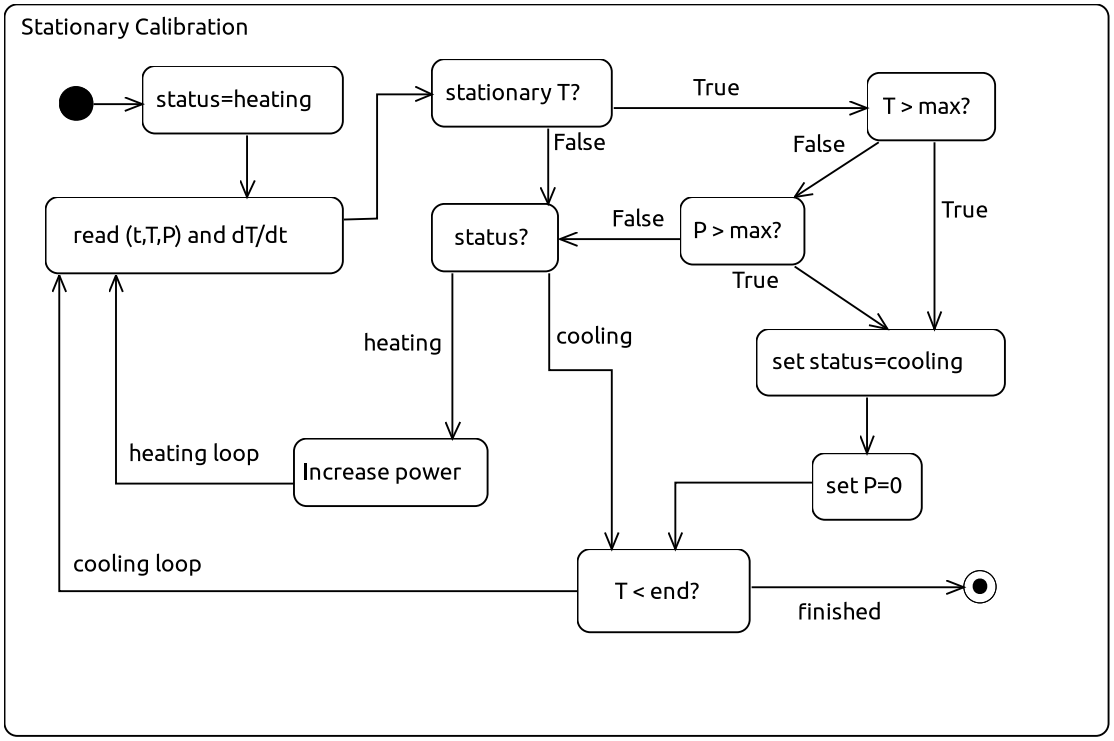
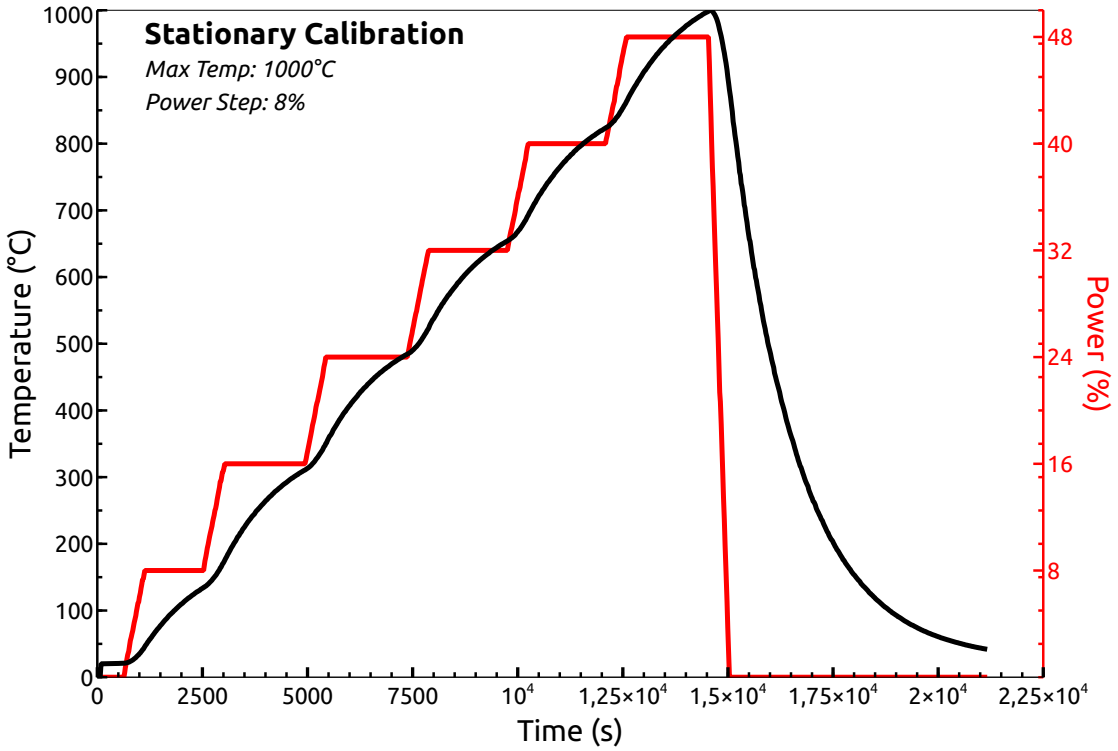


Figure 6.5: Stationary Calibration Curve



$$\frac{\partial Q}{\partial T} = C \tag{6.4}$$

$$P_C = \frac{\partial Q}{\partial t} = C \cdot \frac{\partial T}{\partial t} \tag{6.5}$$

By multiplying the heat capacity of the furnace by the target heating rate, we obtain the needed *capacitive* power input value. At this value we should sum dissipative term, the power needed to maintain the target temperature:

$$P(T) = P_E(T) + P_C(T) \tag{6.6}$$

The DCC model comes in by approximating the heat capacity of the furnace, and focuses exclusively on the cooling curve recorded at the end of the stationary calibration.

6.3.2 Newton's Law of Cooling

Newton's Law of Cooling describes, in a simplified way, the convection-cooling of an object immersed in a uniform fluid. For the model to be applied with satisfactory results, there should be a constant heat transfer coefficient between the object and the fluid. Other heat transfer modes should be negligible: conduction, radiation and mass transfer.

This is almost the case of Misura furnaces. Mass transfer never occurs, and the external shield never goes above 200°C. Thus, convection heat transfer coefficient dependency over temperature is minimal, and radiation and conduction are also reduced to a negligible entity, with respect to the total heat capacity of the furnace.

Newton's law states that *the rate of heat loss of a body is proportional to the difference in temperatures between the body and its surroundings:*

$$\frac{dQ}{dt} = \dot{Q} = hA \cdot (T_{env} - T(t)) = -hA \cdot \Delta T(t) = -D\Delta T(t) \tag{6.7}$$

Where:

Q thermal energy of the body, or internal heat;

h heat transfer coefficient;

$T(t)$ temperature of the body;

A surface area of the furnace, in contact with air;

T_{env} environment temperature

$\Delta T(t)$ temperature gradient between internal and environment temperature;

D dissipative (or heat transfer) coefficient, $D = h \cdot A$.

The actual temperature of the body depends on its internal heat is reflected on its temperature value: its heat capacity, C:

$$C = \frac{\partial Q}{\partial T} \tag{6.8}$$

If the heat capacity is constant, we can express this law as:

$$\frac{dT}{dt} = -\frac{D}{C}\Delta T(t) \tag{6.9}$$

We can define the transfer-capacity ratio as:

$$D' = \frac{D}{C} \tag{6.10}$$

Which, given a starting temperature T_0, leads to the solution:

$$T(t) = T_{env} + (T_0 - T_{env})e^{-\frac{D}{C}t} \tag{6.11}$$

Unfortunately the Newton's model cannot be applied as is to the furnaces. Indeed, *which is the temperature of the object*?

Convection happens at the external shield of the kiln, exposed to air, so its equation needs the temperature of the shield.

The instrument only knows the internal temperature of the internal chamber, which may be much higher than the external one (1600°C opposed to 200°C).

Even if the external temperature was measured, we would need to also model the relation between internal and external temperature: the heat transfer between chamber and shield.

6.3.3 The Fourier's Law of Heat Conduction

The heat flow between internal chamber and external shield of the furnace is governed by conduction through the refractory layer protecting the chamber.

Fourier's law states that *the time rate of heat transfer through a material is proportional to the negative gradient in the temperature and to the area, at right angles to that gradient, through which the heat is flowing.*

In its simplest integral form, it is expressed as:

$$\frac{dQ_{ch}}{dt} = -kA_{ch} \cdot \frac{(T_{ch}(t) - T_{sh}(t))}{x} = -K(T_{ch}(t) - T_{sh}(t)) \tag{6.12}$$

Where:

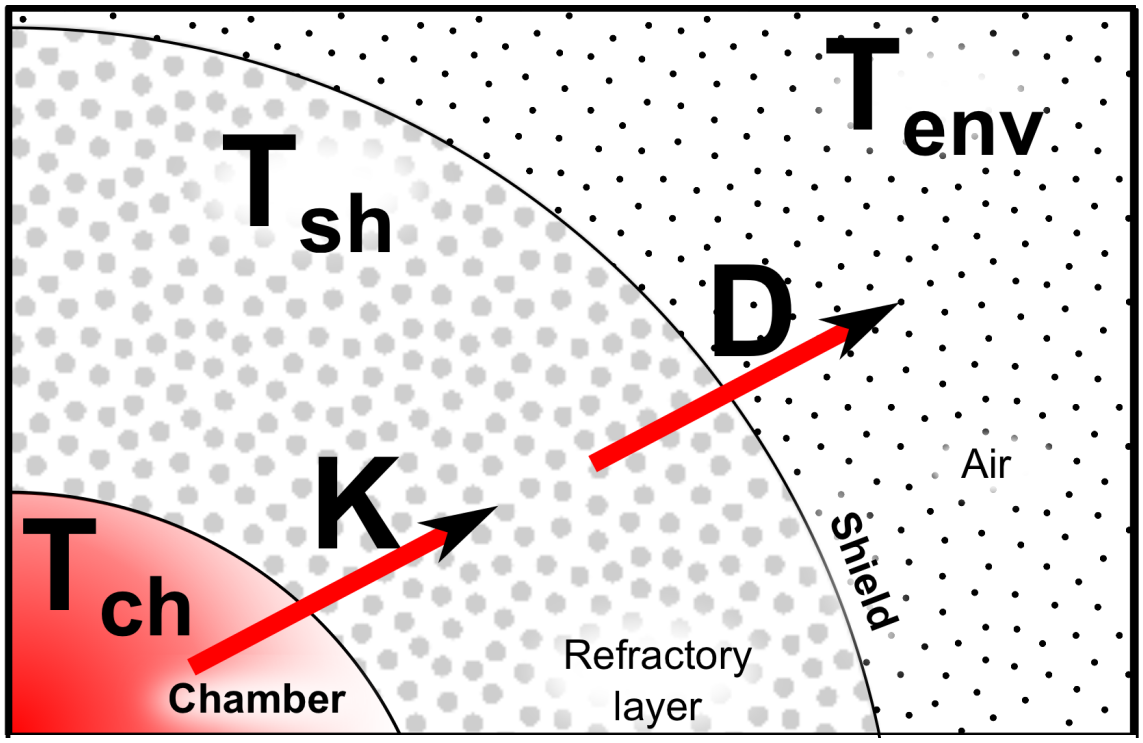
Q_{ch} internal heat of the chamber;

k refractory conductivity;

A_{ch} cross-sectional surface area where the transfer occurs in the refractory;

T_{ch} temperature inside the chamber;

Figure 6.6: Heat transfer from furnace chamber to air



T_{sh} temperature of the shield;

x refractory layer thickness;

K conductive coefficient, $K = \frac{kA_{ch}}{x}$.

Similarly to the Newton's law, if the heat capacity of the internal chamber C_{ch} is constant, eq. 6.12 can also be expressed as:

$$\frac{dT_{ch}}{dt} = -\frac{K}{C_{ch}}(T_{ch}(t) - T_{sh}(t)) \tag{6.13}$$

We can define the transfer-capacity ratio as:

$$K' = \frac{K}{C_{ch}} \tag{6.14}$$

6.3.4 Model Approximation

Summarizing the furnace description, in Figure 6.6:

1. The total heat flowing away from the furnace depends on the temperature of the shield (and of the environment) following Newton's Law of Cooling.

2. The temperature of the shield depends on the temperature of the chamber and the conduction through the refractory, governed by Fourier's Law of Heat Conduction.

The problem can be expressed as a system of ordinary differential equations, by simultaneously considering both Eq. 6.9 and Eq. 6.13:

$$\begin{cases} \frac{dT_{sh}}{dt} = & -\frac{D}{C} \cdot (T_{sh}(t) - T_{env}) \\ \frac{dT_{ch}}{dt} = & -\frac{K}{C_{ch}} \cdot (T_{ch}(t) - T_{sh}(t)) \end{cases} \tag{6.15}$$

Its solution was found using the Sage online mathematical engine [Stein et al., 2012] and Maxima computer algebra system [Maxima, 2009]. The complete form is:

$$\begin{cases} T_{sh}(t) = & T_{env} + (T^0_{sh} - T_{env})e^{-D't} \\ T_{ch}(t) = & T_{env} + \frac{\left\{D' \cdot \left[(T^0_{ch} - T_{env}) - K'(T^0_{ch} - T^0_{sh})\right] \cdot e^{-K't} - K' \cdot (T^0_{sh} - T_{env}) \cdot e^{-D't}\right\}}{D' - K'} \end{cases} \tag{6.16}$$

By defining and substituting some time-zero delta, the second equation becomes:

$$\begin{aligned} \Delta_{ch} &= T^0_{ch} - T_{env} \\ \Delta_{sh} &= T^0_{sh} - T_{env} \\ T^0_{ch} - T^0_{sh} &= \Delta_{ch} - \Delta_{sh} \\ T_{ch}(t) &= T_{env} + \frac{D'(\Delta_{ch} - K'(\Delta_{ch} - \Delta_{sh}))}{D' - K'} \cdot e^{-K't} - \frac{K'\Delta_{sh}}{D' - K'} \cdot e^{-D't} \end{aligned} \tag{6.17}$$

It is important to notice the general form of this apparently complex solution:

$$T_{ch}(t) = T_{env} + W \cdot e^{-K't} - Z \cdot e^{-D't} \tag{6.18}$$

Two exponential decay functions concur in cooling down the chamber of the furnace.

The second decay depends from the dissipative coefficient between the shield of the furnace and the environmental air, and considers only ΔT_{sh} at shield-air interface.

The first decay is determined by the conductivity of the refractory layer, insulating the internal chamber. It considers both the total temperature delta between chamber and air ΔT_{ch}, and between shield and air.

6.3.4.1 Cooling curve fitting

The solution in eq. 6.17 is fitted against the real cooling behavior of the furnace, recorded at the end of the stationary calibration. The known parameters of the fitting are:

- all $(t, T_{ch}(t))$ points constituting the cooling curve
- T_{env} corresponding to $T_{ch}(0)$
- T^0_{ch} is the maximum temperature at which the cooling curve begins
- thus, Δ_{ch} is known

Parameters being fitted are:

- $D' = \frac{D}{C_{sh}}$, dissipative factor over capacitive constant of the shield
- $K' = \frac{K}{C_{ch}}$, conductive factor over capacitive constant of the chamber and the refractory layer
- T_{sh}^0, the temperature of the shield at the beginning of the cooling curve (or the equivalent Δ_{sh})

The fitting is performed using the quasi-Newton method of Broyden, Fletcher, Goldfarb, and Shanno (BFGS). The method optimally requires the partial derivatives of eq. 6.17 towards the fitted parameters, which can be analytically computed:

$$\begin{cases} \frac{\partial T_{ch}}{\partial D'} = & \frac{1}{\Lambda}\left[G \cdot t \cdot e^{-D't} - F \cdot e^{-Kt} + \frac{F \cdot D' \cdot e^{-K't} - G \cdot e^{-D't}}{\Lambda}\right] \\ \frac{\partial T_{ch}}{\partial K'} = & \frac{1}{\Lambda}\left[F \cdot D' \cdot t \cdot e^{-K't} - D'(\Delta_{ch} - \Delta_{sh}) \cdot e^{-K't} - \Delta_{sh} e^{-D't} - \frac{F \cdot D' \cdot e^{-K't} - G \cdot e^{-D't}}{\Lambda}\right] \\ \frac{\partial T_{sh}^0}{\partial \Delta_{sh}} = & \frac{1}{\Lambda}\left[D'K'e^{-K't} - K'e^{-D't}\right] \end{cases} \tag{6.19}$$

Where:

$$\begin{aligned} \Lambda &= D' - K' \\ F &= K'(\Delta_{ch} - \Delta_{sh}) - \Delta_{ch} \\ G &= K'\Delta_{sh} \end{aligned}$$

6.3.4.2 Heat Capacity Approximation

The cooling fitting should return reasonable D' and K' values. Recalling their definition (eq. 6.10 and 6.14), they are a proportion between a heat conduction factor and a heat capacity of the two parts composing the furnace model (shield and chamber, Figure 6.6).

In order to determine the input heat flux needed to obtain a certain heating rate, the heat capacity of the body being heated is required.

The target temperature of the process control is T_{ch}, where the sample is placed: so, the body being considered is the chamber. Knowing the refractory material's conductivity k, thickness x and area of transfer A_{ch}, its conductive factor $K = \frac{kA_{ch}}{x}\, C_{ch}$ could be determined. Then, $C_{ch} = \frac{K}{K'}$.

But this information can also be extrapolated from the behavior of the calibration curve, recalling the stationary power interpolating function definition $P_E(T)$.

The stationary power interpolating function $P_E(T)$ returned the power values needed to maintain temperature T, $\frac{dT}{dt} = 0$. Applied to refractory heat transfer equation 6.12, it means at a certain temperature the chamber's input/output flows are equal:

$$\begin{aligned} P_E &= -\frac{dQ_{ch}}{dt} \\ P_E &= K(T_{ch}(t) - T_{sh}(t)) \\ K &= \frac{P_E}{T_{ch}(t) - T_{sh}(t)} \end{aligned} \tag{6.20}$$

Equation 6.20 contains the $T_{sh}(t)$ term, the temperature of the shield at any given time. If this value is not measured someway, we should accept the following approximation:

The chamber response to input power is fast enough for the shield convective cooling effect to be neglected: $T_{sh}(t) = T_{env}$

The heat capacity equation can now be substituted in the DDC output power equation 6.6:

$$P(T) = P_E(T) + \frac{P_E(T)}{(T - T_{env})} \cdot \frac{1}{K'} \tag{6.21}$$

As the convective effect was disregarded, D' factor is not present. Anyway, if the cooling curve was fitted without considering this effect, it would have lead to a wrong K' value.

The convective effect is not totally disregarded. On the contrary it is *overestimated* as the chamber temperature increases. Indeed, while $T_{sh} = T_{env}$ approximation is exact at the beginning of a cycle, it will start differing as the real T_{sh} increases. The theoretical power should then decrease according to eq. 6.20, but the approximated power will be higher by an error e_P:

$$e_P = P_E \frac{T_{sh} - T_{env}}{(T - T_{env})(T - T_{sh})} \tag{6.22}$$

The error is directly proportional to the product between the stationary power and the error introduced by approximating $T_{sh} = T_{env}$. But is inversely proportional to the product of the delta between chamber temperature, environment and shield.

This error is well-behaved:

- The equilibrium power will remain between 0-100%
- $T_{sh} - T_{env}$ will never rise above 200°C
- T_{sh} rise is much slower than T

The numerator will increase very slowly and up to a maximum value of $100 \cdot 180 = 1.8 \cdot 10^4$ when $P_E = 100$, $T_{sh} = 200$, T_{env}=20.

The denominator rises very quickly and up to a maximum value, for $T = 1600$, around $1520 \cdot 1400 \cong 2 \cdot 10^6$. This leads to $e_P = 8.4 \cdot 10^{-3}$, which is not even *measurable* on most power unit controller device, usually having a resolution of 0.1%.

6.3.5 Defects and Advantages of DDC regulator

The bigger DDC defect was already been explained while introducing it. Being an *open-loop* controller, it cannot react to external, unpredictable events. For example, sample's phase transitions, endo/exothermic reactions or changes in laboratory conditions.

This is also the main feature of DDC: it will never behave in an unpredictable or dangerous way, given the calibration was done correctly.

Furnace modeling introduces many approximations, for example it underestimates the shield temperature (eq. 6.21). As a result, the DDC output power will always be overestimated by a certain amount.

But the biggest advantage of DDC is its reactivity: unlike PID, it does not need an error to be accumulated before approximating a good power output regime.

If the requested heating rate requires 10% of power in order to be realized from a starting temperature of 20°C, the full 10% is *immediately* emitted.

DDC will not eliminate the physical delay in temperature rise due to the physical heat transfer process, but will at least mitigate the mathematical artifact introduced by the PID control.

6.4 Predictive Regulator

The `Predictive` regulator takes the advantages of both PID and DDC algorithms.

Both concur in determining the total power output at any given time and temperature, by means of a weighted sum:

$$\begin{aligned} P &= w_{PID} \cdot P_{PID} + w_{DDC} \cdot P_{DDC} \\ 1 &= w_{PID} + w_{DDC} \end{aligned}$$

For the algorithm to be useful, $w_{DDC} > w_{PID}$, practically $w_{DDC} = 0.8$, $w_{PID} = 0.2$. In these conditions the DDC algorithm determines most of the power output, but leaves to the PID regulator the possibility to correct the final output by at most 20%.

The `Predictive` regulator reduces the instability of PID to a maximum 20% of the total power throughput, while retaining 80% of DDC reactivity in setpoint rate changes.

If a deviation from the model is needed to reach the setpoint, it will be emitted up to 20% of its entity, as it would not be safe for the furnace and the heating element to receive more than this power.

Chapter 7

Conclusions

The new image analysis and thermal control algorithms discussed in this thesis were tested against a small set of available materials. Their effectiveness should be now demonstrated by an intense experimental research activity, comprising the most disparate samples and with a rigorous comparison with international standards and current scientific literature.

The paradigm shift introduced in the instrument control software enables new kind of experiments to be performed. For example, with arbitrarily complex thermal cycles, or extreme samples positioning (inclined planes, moving drops, etc). New applications should be extensively documented and proved to be insightful.

The Misura 4 software is starting its internal testing phase (alpha), and the development will continue following different branches. Image analysis will be extended with:

- Image superresolution for combining multiple frames into a single, higher resolution image [Zalevsky and Mendlovic, 2004]. This could allow to break the $0.5\mu m$ limitation optical instruments have.
- Surface tension determination, approximating the fully-augmented Young-Laplace equation [DasGupta et al., 1993] with an high-magnification optic (combined with superresolution).
- The fleximetry curve of a multilayer material may be fitted against expansion curves of single layers, inferring layers properties like elasticity and coupling temperature between layers.
- Extended simulation capability. Simulation of devices and behavior was mainly implemented for unit-testing and demonstration purposes, but is now easily extensible towards real, complete materials simulation. For example, perfect visco-elastic pyro-plastic deformation is already being simulated. A simplified smoothed particle hydrodynamics model of a sample softening is currently under study [Liu and Liu, 2003].

Bibliography

[Alted et al., 02] Alted, F., Vilata, I., et al. (2002--). PyTables: Hierarchical datasets in Python. 2.2.1

[Anonymous, 2002] Anonymous (2002). 1:61--62. 1.1

[Apache Software Foundation, 2012] Apache Software Foundation (2012). Apache subversion. enterprise-class centralized version control for the masses. 2.3.1

[Bernardin et al., 2006] Bernardin, A. M., de Medeiros, D. S., and Riella, H. G. (2006). Pyroplasticity in porcelain tiles. *Materials Science and Engineering: A*, 427(1-2):316 -- 319. 1.3.1.1

[Bertolino, 2007] Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering*, FOSE '07, pages 85--103, Washington, DC, USA. IEEE Computer Society. 2.3

[Bitzer and Schroder, 2005] Bitzer, J. and Schroder, P. J. (2005). The impact of entry and competition by open source software on innovation activity. Industrial Organization 0512001, EconWPA. 2.2.1

[Black Duck Software, 2011] Black Duck Software (2011). Ohloh. your guide to open source. 2.1.1

[Boshernitsan et al., 2006] Boshernitsan, M., Doong, R., and Savoia, A. (2006). From daikon to agitator: lessons and challenges in building a commercial tool for developer testing. In *Proceedings of the 2006 international symposium on Software testing and analysis*, ISSTA '06, pages 169--180, New York, NY, USA. ACM. 2.3

[Bradski, 2000] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*. 2.2.1

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media Inc. 2.2.1

[Brent, 1973] Brent, R. P. (1973). *Algorithms for Minimisation Without Derivatives*. Prentice Hall. 5.4.1.2

[Brun et al., 2011] Brun, Y., Holmes, R., Ernst, M. D., and Notkin, D. (2011). Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 168--178, New York, NY, USA. ACM. 2.3, 2.3.1

[C. Özgüra, 2011] C. Özgüra, O. S. (2011). Sintering effect on the microstructure of glassy porous ceramics. In *6th International Advanced Technologies Symposium*. 1.2.2.6

[CEA LIST, 2012] CEA LIST, Atos Origin, L. (2012). Papyrus UML modeling for eclipse. 2.3.4

[Chiara Venturelli, 2005] Chiara Venturelli, M. P. (2005). Study of the sintering behaviour of some multilayer advanced ceramics by means of the optical non-contact dilatometry. *CFI. Ceramic forum international*, 82(5). 1.2.2.6

[Chiara Venturelli, 2006] Chiara Venturelli, M. P. (2006). Thermo-mechanical behaviour of technical ceramic bricks, felt boards and fibrer. *CFI. Ceramic forum international*, 84(4). 1.2.2

[DasGupta et al., 1993] DasGupta, S., Schonberg, J. A., Kim, I. Y., and Jr., P. C. W. (1993). Use of the augmented young-laplace equation to model equilibrium and evaporating extended menisci. *Journal of Colloid and Interface Science*, 157(2):332 -- 342. 7

[Davide Sighinolfi, 2012] Davide Sighinolfi, C. V. (2012). Experimental study of deformations and state of tension in traditional ceramic materials. *Tiles and Bricks International*, 1. 1.3.1.1

[Davide Sighinolfi, 2009] Davide Sighinolfi, M. P. (2009). Effect of quartz particle size on porcelain stoneware sintering by means of optical dilatometry. *CFI. Ceramic forum international*, 86(3). 1.2.2.2

[Davide Sighinolfi, 2010a] Davide Sighinolfi, M. P. (2010a). Optical dilatometry / tga investigations on glass-ceramics containing iron-rich slug from steel industries. *CFI. Ceramic forum international*, 87(8-9). 1.2.2.1

[Davide Sighinolfi, 2010b] Davide Sighinolfi, M. P. (2010b). Sintering of metallic powders on ceramic substrates. *CFI. Ceramic forum international*, 87(11-12). 1.2.2.7

[Davide Sighinolfi, 2010c] Davide Sighinolfi, M. P. (2010c). Thermal expansion investigation on glasses used in ceramic industry. *CFI. Ceramic forum international*, 87(6-7). 1.2.2

[E. Eren and Ay, 2007] E. Eren, A. K. and Ay, N. (2007). An investigation into phase and microstructural evolution of zinc oxide containing porcelain bodies. In *10th International Conference and Exhibition of the European Ceramic Society*. European Ceramic Society. 1.3.1.1

[Eclipse, 2012] Eclipse (2012). Eclipse modeling platform. 2.3.4

[Edgewall, 2011] Edgewall (2011). Trac integrated scm and project management. 2.3.3

[Elsevier B.V., 2012] Elsevier B.V. (2012). ScienceDirect. 2.2.1

[Fernanda Andreola and Lancellotti, 2010] Fernanda Andreola, L. B. and Lancellotti, I. (2010). Recovery of glazing ceramic sludge in construction materials. In *Second International Conference on Sustainable Construction Materials and Technologies*. 1.2.2.4

[Flugel et al., 2007] Flugel, A., Dolan, M. D., Varshneya, A. K., Zheng, Y., Coleman, N., Hall, M., Earl, D., and Misture, S. T. (2007). Development of an improved devitrifiable fuel cell sealing glass. *Journal of The Electrochemical Society*, 154(6):B601--B608. 1.1.2.7

[Freier et al., 1996] Freier, A. O., Karlton, P., and Kocher, P. C. (1996). The ssl protocol --- version 3.0. Internet Draft, Transport Layer Security Working Group. 4.1

[Frigeri and Speranza, 2011] Frigeri, A. and Speranza, G. (2011). "eppur si muove", software libero e ricerca riproducibile. *CoRR*, abs/1105.1885. 2.2.1

[Fröberg, 2007] Fröberg, L. (2007). *Factors affecting raw glaze properties*. Åbo Akademi University, Process Chemistry Centre. 1.1.2.1

[Gamma et al., 1994] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition. 4.2.1

[Georg von Krogh, 2007] Georg von Krogh, S. S. (2007). The open source software phenomenon: Characteristics that promote research. *Journal of Strategic Information Systems*. 2.2.1

[Google, 2012] Google (2012). Google Scholar. 2.2.1

[Greil, 1989] Greil, P. (1989). Processing of silicon nitride ceramics. *Materials Science and Engineering: A*, 109(0):27--35. Proceedings of the Symposium on Ceramic Materials Research. 6

[GStreamer, 2012] GStreamer (2012). GStreamer Open Source Multimedia Framework. 5.1.1

[Holzworth et al., 2011] Holzworth, D. P., Huth, N. I., and deVoil, P. G. (2011). Simple software processes and tests improve the reliability and usefulness of a model. *Environmental Modelling and Software*, 26(4):510 -- 516. (document), 2.3, 2.4, 2.3.2

[Ibanez, 2011] Ibanez, L. (2011). Default to open: The scientific method. 2.2.1

[Jain, 1989] Jain, A. (1989). *Fundamentals of digital image processing*. Prentice-Hall information and system sciences series. Prentice Hall. 5.2.1

[Kaisersberger et al., 1988] Kaisersberger, E., Janoschek, J., and Hädrich, W. (1988). Thermal analysis in the field of high temperature superconductors. *Thermochimica Acta*, 133(0):43 -- 48. 6

[Klegues et al., 2009] Klegues, O. R., Joaquim, F. F., de, O. F. J., ElÃdio, A., and Montedo, B. A. M. (2009). Sintering behavior of lzsa glass-ceramics. *Materials Research*, 12:197--200. 1.2.2.3, 1.2.2.4

[Linux Foundation, 2010] Linux Foundation (2010). Linux adoption trends: A survey of enterprise end users. 2.2.1

[LinuxTv, 2011] LinuxTv (2011). Linux media infrastructure api. 5.1.1

[Liu and Liu, 2003] Liu, G. and Liu, M. (2003). *Smoothed particle hydrodynamics: a meshfree particle method*. World Scientific. 7

[Lutz and Prechelt, 2003] Lutz and Prechelt (2003). Are scripting languages any good? a validation of perl, python, rexx, and tcl against c, c++, and java. volume 57 of *Advances in Computers*, pages 205 -- 270. Elsevier. 2.2.1

[Maxima, 2009] Maxima (2009). Maxima, a computer algebra system. version 5.18.1. 6.3.4

[Mazey, 1946] Mazey, E. (1946). Duck test (proverb). 3.3.2

[McCollor, 1997] McCollor, C. J. Z. D. P. (1997). Ash behavior in power systems. Report for the U.S. Department of Energy. 1.1.2.6

[Mushtaq Ahmed, 2003] Mushtaq Ahmed, D. A. E. (2003). Characterising glaze-melting behavior via hsm. *American Ceramic Society Bulletin*, 81(3). 1.1.2.1

[Nelson, 2007] Nelson, E. (2007). Results of the visual basic survey: Part 2 visual basic 6.0 usage today. 2.1.1

[Nezahat Ediza, 2009] Nezahat Ediza, A. Y. (2009). Characterization of porcelain tile bodies with colemanite waste added as a new sintering agent. *Journal of Ceramic Processing Research*, 10(4):414--422. 1.2.2.4

[Nvidia Corporation, 2007] Nvidia Corporation (2007). Cuda. 2.2.1

[OMG, 2011] OMG (2011). Uml - unified modeling language by object management group. 2.3.4

[Osman ana, 2008] Osman ana, Cem Özgüra, C. K. (2008). Preparation and characterization of a highly hydrophilic glassy pore wall membrane filter. *Journal of Ceramic Processing Research*, 9(3):296--301. 1.1.2.4

[Osterweil and Clarke, 1990] Osterweil, L. and Clarke, L. A. (1990). Directions for u.s. research and development efforts on software testing and analysis. Technical report, Amherst, MA, USA. 2.3

[Özgüra O., 2007] Özgüra O., Sana* C., K. C. (2007). Preparation of high performance capillary ceramic filter using frit glass powders. In *10th International Conference and Exhibition of the European Ceramic Society*. European Ceramic Society. 1.1.2.4

[Paganelli, 2010] Paganelli, D. (2010). Ticket #489. memory leak in cv.moments, cv.polyline. 2.2.1.1

[Paganelli, 2002] Paganelli, M. (2002). Using the optical dilatometer to determine sintering behavior. *American Ceramic Society Bulletin*, 81(11):25--30. 1.2.2.2

[Paganelli, 2003a] Paganelli, M. (2003a). The measurement of surface tension at high temperatures on glass samples using the drop profile analysis. *The American Ceramic Society Bulletin*, 82(9). 1.1.2.8

[Paganelli, 2003b] Paganelli, M. (2003b). Measuring thermal expansion of ultra-thin samples. *Ceramic Industry*. 1.2.2

[Paganelli, 2004] Paganelli, M. (2004). Double beam optical dilatometry: A novel tool for the ceramic research. *CFI. Ceramic forum international*, 81(6-7). 1.2

[Palmour et al., 1974] Palmour, H., Palmour, H., Hare, T., Huckabee, M., DIV., N. C. S. U. R. E. R. S., and Command, U. S. N. A. S. (1974). *Optimal Densification of Ceramics by Rate Controlled Sintering*. Defense Technical Information Center. 6

[Pekkan and Karasu, 2009] Pekkan, K. and Karasu, B. (2009). Production of opaque frits with low zro and zno contents and their industrial uses for fast single-fired wall tile glazes. *Journal of Materials Science*, 44:2533--2540. 10.1007/s10853-009-3329-7. 1.1.2.2

[Phillip Merrick, 2006] Phillip Merrick, S. A. J. L. (2006). A remote procedure call (RPC) uses a message expressed in a mark-up language message encoding, with type labels indicating data item types. 2.4

[Postolenko, 2008] Postolenko, V. (2008). Failure mechanisms of thermal barrier coatings for high temperature gas turbine components under cyclic thermal loading. 1.3.1.2

[Project, 2012] Project, T. Q. (2012). *Model/View Programming*. `http://qt-project.org/doc/qt-4.8/model-view-programming.html`. 4.2.1

[Python Software Foundation,] Python Software Foundation. 3.1

[Python.org, 2012] Python.org (2012). Unit testing framework - python standard library. 2.3.2

[R. de Gennaroa, 2009] R. de Gennaroa, A. Langellab, M. D. M. D. A. C. P. C. M. d. G. (2009). Use of zeolite-rich rocks and waste materials for the production of structural lightweight concretes. *Environmental Science and Technology*, 43:7123--7129. 1.2.2.4

[Ragulya and Skorokhod, 1995] Ragulya, A. and Skorokhod, V. (1995). Rate-controlled sintering of ultrafine nickel powder. *Nanostructured Materials*, 5(7-8):835 -- 843. 6

[Ragulya and Skorokhod, 1998] Ragulya, A. and Skorokhod, V. (1998). Validity of rate-controlled sintering of dense nanocrystalline materials. *Metal Powder Report*, 53(7-8):45 --. 6

[Reese, 2000] Reese, G. (2000). *Distributed Application Architecture*. O'Reilly & Associates., 2 edition. 2.2.1

[Ricerche e Studi S.p.A., 2011] Ricerche e Studi S.p.A., M. S. U. S. (2011). Indagine annuale sulle principali imprese multinazionali del mondo (luglio 2011). 2.4

[Rivest, 1992] Rivest, R. L. (1992). The md5 message-digest algorithm. Internet RFC 1321. 4.1

[Roff, 2003] Roff, J. (2003). *Fondamenti di UML. Con CD-ROM*. McGraw-Hill Companies. 2.3.4

[Saff and Ernst, 2004] Saff, D. and Ernst, M. D. (2004). An experimental evaluation of continuous testing during development. *SIGSOFT Softw. Eng. Notes*, 29:76--85. 2.3

[Sanders, 2012] Sanders, J. (2012). Veusz - a scientific plotting package. 2.2.1, 2.6.4

[Schabbach et al., 2011] Schabbach, L. M., Andreola, F., Karamanova, E., Lancellotti, I., Karamanov, A., and Barbieri, L. (2011). Integrated approach to establish the sinter-crystallization ability of glasses from secondary raw material. *Journal of Non-Crystalline Solids*, 357(1):10 -- 17. 1.2.2.4

[Serra, 1983] Serra, J. (1983). *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA. 5.2.1

[Sighinolfi and Paganelli, 2009] Sighinolfi, D. and Paganelli, M. (2009). New double optic heating microscope for multi-standard analysis for solid fuel ashes. *cfi/Ber DKG*, 86(5). 1.1.2.6

[Sighinolfi and Paganelli, 2011] Sighinolfi, D. and Paganelli, M. (2011). Study of the behaviour of mould powders for continuous casting by using the heating microscope. *IOP Conference Series: Materials Science and Engineering*, 18(22):222007. 1.1.2.5

[Silvano Pagliuca, 2011] Silvano Pagliuca, W. d. F. (2011). Porcelain enamel and industrial enamelling process. 1.1.2.3

[Speyer et al., 1992] Speyer, R. F., Echiverri, L., and Lee, C. K. (1992). A shrinkage rate-controlled sintering dilatometer. *Journal of Materials Science Letters*, 11:1089--1092. 10.1007/BF00730840. 6

[Stein et al., 2012] Stein, W. et al. (2012). *Sage Mathematics Software*. The Sage Development Team. `http://www.sagemath.org`. 6.3.4

[Summerfield, 2007] Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt*. Prentice Hall. 2.2.1, 2.2.1

[Suzuki and Abe, 1985] Suzuki, S. and Abe, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32--46. 5.2

[The HDF Group, 2010] The HDF Group (2010). Hierarchical data format version 5. 2.2.1

[TOP500.org, 2011] TOP500.org (2011). Operating system share over time. 2.2.1

[Torkar, 2005] Torkar, R. (2005). A literature study of software testing and the automated aspects thereof. Technical report, P.O. Box 957, SE-461 29 TrollhÃ€ttan, Sweden. 2.3

[Uhlig et al., 2011] Uhlig, F., Al-Turany, M., Bertini, D., and Karabowicz, R. (2011). Software development infrastructure for the fair experiments. *Journal of Physics: Conference Series*, 331(4):042024. 2.3.3

[UNU-MERIT, 2006] UNU-MERIT, E. (2006). Economic impact of open source software on innovation and the competitiveness of the information and communication technologies (ict) sector in the eu. 2.2.1

[Uwe et al., 2007] Uwe, E., Boccaccini, A., Cook, S., and Cheeseman, C. (2007). Effect of borate addition on the sintered properties of pulverised fuel ash. *Ceramics International*, 33(6):993 -- 999. 1.2.2.5

[VDC Research, 2011] VDC Research (2011). 2011 embedded software & tools market intelligence service. 2.2.1

[Venturelli, 2011] Venturelli, C. (2011). Heating microscopy and its applications. *Microscopy Today*, 19(01):20--25. 1.1.2.5

[Wikipedia, 2011a] Wikipedia (2011a). Usage share of operating systems. 2.2.1

[Wikipedia, 2011b] Wikipedia (2011b). XML-RPC. 2.4

[Wikipedia, 2012a] Wikipedia (2012a). Object-oriented programming --- Wikipedia, the free encyclopedia. 2.1.1

[Wikipedia, 2012b] Wikipedia (2012b). World wide web --- Wikipedia, the free encyclopedia. [Online; accessed Gen 2012]. 2.4

[Wikipedia, 2012c] Wikipedia (2012c). World wide web consortium --- Wikipedia, the free encyclopedia. [Online; accessed Gen 2012]. 2.4

[William M. Carty, 2003] William M. Carty, Ph.D. Christopher W. Sinton, P. (2003). Selective batching for improved commercial glass melting. Report for the U.S. Department of Energy. 1.1.2.4

[Zalevsky and Mendlovic, 2004] Zalevsky, Z. and Mendlovic, D. (2004). *Optical superresolution*. Springer series in optical sciences. Springer. 7

[Zhang et al., 2004] Zhang, D., Kamel, M., and Baciu, G. (2004). *Integrated image and graphics technologies*. The Kluwer international series in engineering and computer science. Kluwer Academic. 5.4.2.2

[Zhang et al., 2009] Zhang, D., Vedel, E., Hupa, L., Aro, H. T., and Hupa, M. (2009). Predicting physical and chemical properties of bioactive glasses from chemical composition.part 3: In vitro reactivity. *Glass Technology - European Journal of Glass Science and Technology Part A*, 50(1):1--8. 1.1.2.4

[Zymaris, 2009] Zymaris, C. (2009). What open source shares with science. 2.2.1