UNIVERSITEIT GENT

Ghent University
Faculty of Sciences
Department of Applied Mathematics and Computer Science
Department of Biochemistry and Microbiology

StrainInfo: from microbial information to microbiological knowledge

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Bert Verslyppe

July 2012

Promotors: Prof. Dr. Peter Dawyndt
Prof. Dr. Bernard De Baets
Prof. Dr. Paul De Vos

Contents

Acknowledgements

I wish to thank my advisor Prof. Dr. Peter Dawyndt for generating my interest in this subject and for providing excellent guidance during my trajectory towards this doctoral dissertation. My co-advisors Prof. Dr. Paul De Vos and Prof. Dr. Bernard De Baets are strongly appreciated for providing their different perspective and therefore valuable feedback on my papers and research.

A special thanks goes to my Ph.D. colleague Drs. Wim De Smet for the fruitful collaboration which resulted in the StrainInfo platform as it is today, and I wish him all the best while writing his dissertation. I want to thank Wim Gillis for his contributions in the development and administration of the StrainInfo platform. Furthermore, I want to thank Tom De Groote and Bert Sarens for their assistance in the administration of all infrastructure, and our interns Dieter Mourisse and Kenneth Waegeman for their successful projects.

Some of the work presented in this dissertation is the result of study leaves abroad. I want to thank Prof. Enrico Motta, Ph.D. and Andriy Nikolov, Ph.D. from the Knowledge Media institute (KMi) of The Open University in Milton Keynes, UK for welcoming me in their dynamic group for an internship in December 2009. Prof. Eduard Hovy, Ph.D., Gully Burns, Ph.D. and Cartic Ramakrishnan, Ph.D. from the USC Information Sciences Institute in Marina del Rey, Los Angeles, CA, USA are thanked for the very interesting period that my stay in their group has been (September 2010–July 2011).

I thank Tom Dedeurwaerdere for providing information regarding the ITPGRFA. Gully Burns for his helpful comments on the manuscript that forms the basis for Chapter 4. Figure 4.5 was designed by Form Apparatus. Claudine Vereecke and the BCCM/LMG collection for their valuable input from the BRC perspective and their help in resolving the history of the LMG 4090 culture. Furthermore, I want to thank the numerous volunteers from the Laboratory of Microbiology as well as international researchers for their valuable curation efforts in the Make Histri project. MCL was developed in collaboration with Dr. Renzo Kottmann (Microbial Genomics Group, MPI for Marine Microbiology, Bremen, Germany) in the context of the Genomic Standards Consortium (October 2009). In addition, I want to thank the MCL early adopters who have made their catalog available in XML and provided feedback on the language structure.

Last but not least, I also want to express my appreciation for the large and dedicated user community of StrainInfo. By requesting features or pointing out inaccuracies, their continuous feedback helps to continue to improve StrainInfo and shape the future of microbiological research.

The development of the StrainInfo platform (including the Make Histri project and MCL) is funded by the Belgian Science Policy (contract IUAP VI/06). Semastri and Muurge were developed during a study leave at the University of Southern California (USC) Information Science Institute (ISI) funded by the Research Foundation Flanders (FWO reference V.4.246.10.N.01).

Dankwoord

I want to thank the people who contributed indirectly to this dissertation in the language they understand the best. Therefore, most of the remainder of this section continues in Dutch.

Het schrijven van een dankwoord is meestal een van de laatste zaken die een doctoraatsstudent in het kader van zijn doctoraat doet, en voor mij is het niet anders. Dit het moment waar je achter je kijkt, en ziet dat langs de weg die je hebt afgelegd heel wat supporters staan. Zonder de steun van jullie was het onmogelijk om tot deze verhandeling te komen. Ik wil jullie allemaal bedanken voor jullie bijdrage, direct of indirect, bewust of onbewust.

Alles begint bij mijn ouders; zonder hen had ik hier niet gestaan. Jullie hebben een schitterende omgeving om in op te groeien gecreëerd, me bijzonder veel kansen geboden, en – voor informatici ook wel bepalend – geïnvesteerd in computerhardware en internetverbindingen. Ik kreeg van jullie het vertrouwen en kon deze kansen ongelimiteerd benutten. Bedankt. Ik kan moeilijk uitdrukken hoe erkentelijk ik jullie hiervoor ben. Hetzelfde geldt ook voor mijn zussen, Leen en Lies. Het is schitterend om jullie als zussen te mogen hebben en ik ben jullie heel dankbaar voor de leuke sfeer die jullie brengen. En sorry omdat ik vroeger altijd de computer innam en allerlei excuses had waarom ik op extra computertijd aanspraak zou maken...

Mijn familie (de families Vandenberge en Verslyppe) is erg belangrijk voor me. Het is spijtig genoeg onmogelijk om iedereen individueel te bedanken, maar eigenlijk verdienen jullie dat allemaal. In het bijzonder wil ik mijn doopmeter (Tante Maria) en Nonkel Guido bedanken. Tijdens de vele 'TM-kampen' bij julie in Lommel heb ik erg veel bijgeleerd over techniek, electriciteit en wetenschappen. Hier zette ik in augustus 1995 mijn eerste stappen in de informatica met het gebruik van de applicaties Write en Paint in Windows 3.11. Met Nieuwjaar die erop volgde kreeg ik een boek over SuperLogo van jullie; het werd mijn eerste programmeertaal, iets wat ik als een mijlpaal beschouw. In de zomer van 1996 leerde ik om schijven te partitioneren, formatteren en Windows te herinstalleren ("je mag alles doen met

onze computer als je ervoor zorgt dat alles weer in orde is als je terug naar huis gaat"). Jullie hebben mijn talent voor informatica ontdekt, en me ook de eerste kansen gegeven om dat te ontwikkelen. Dank u wel. Mijn neef Bart speelt hierin ook een belangrijke voorbeeldrol. Toen ik mijn eerste informaticastappen aan het zetten was, was jij reeds bezig met je doctoraat. Al sinds toen sta je altijd klaar voor raad en daad: in de beginjaren heb je ons vaak gedepaneerd bij technische problemen die ik veroorzaakt had. Doorheen de jaren is die raad geëvolueerd. Ik apprecieer onze gesprekken vanuit je levens- en technische ervaring heel erg sterk. Ook wil ik je, Marleen, Robin, Anaïs en An bedanken voor de schitterende reizen die we samen gemaakt hebben.

De Bende; Christophe, Maarten & Julia, Bart, Bieke, Ilse, Annelies, Toon, Brecht, Jochen en Martijn: jullie zijn allemaal de max! Ik was eerst een jaar afwezig en toen ik terugkwam, had ik al snel weinig tijd. Ik vind dat ik jullie een beetje verwaarloosd heb en hoop dat ik vanaf nu terug meer tijd met jullie kan doorbrengen. Het wandel- en activiteitenclubje gaat nu dus eindelijk binnenkort van start! Christophe wil ik ook nog even extra bedanken om me voor te stellen aan het meisje dat me tijdens het schrijven van deze verhandeling soep zou leren maken. Kirsten, bedankt om me soep te leren maken – de laatste hoofstukken van deze verhandeling werden geschreven bij een dieet van zelfgemaakte bloemkool-broccoli-soep – en om je fantastische zelf te zijn. Ik beloof dat we nu écht naar Technopolis gaan.

Tijdens mijn studie informatica werd de basis voor dit doctoraat gelegd, en de mensen die ik toen leerde kennen zijn bepalend voor het welslagen hiervan. Ik denk bijvoorbeeld aan Bart, Davy, Femke D., Femke O., Jeroen F., Jeroen J., Pascal en Stéphanie. De gezelligheid van tijdens de les zette zich voort tijdens onze 'informaticameetings'. De meesten van ons zijn aan een doctoraat begonnen; sommigen hebben de eindmeet reeds gehaald, de anderen wens ik een vlotte schrijfperiode toe. Ik hoop dat we nog veel informaticameetings kunnen organiseren! Verder wil ik ook de voltallige Lucasbuilding, en Liesbeth in het bijzonder, bedanken voor de leuke tijd op kot.

Ik had de kans om met twee verschillende vakgroepen samen te werken en begeleid te worden door drie promoteren. Ik wil m'n promotoren Peter Dawyndt, Paul De Vos en Bernard De Baets bedanken voor hun steun en feedback, elk vanuit hun eigen invalshoek. Aan de vakgroep Toegepaste Wiskunde en Informatica heb ik het geluk gehad om op een gezellige bureau te belanden. Ik bracht er heel veel tijd door met Heide, Patricia en Veerle: bedankt voor de leuke tijd, de steun en de interessante gesprekken en nieuwtjes. De sfeer aan de vakgroep wordt (en werd)

bepaald door het enthousiasme van de fantastische collega's (en ex-collega's die de vakgroep spijtiggenoeg ondertussen verlaten hebben): oa. Adriaan, Bart, Catherine, Charlotte, Davy, Gilles, Glad, Heide, Hilde, Jan (2x), Jeroen, Karel (2x), Michael, Nele (2x), Patricia, Stéphanie, Steven, Tom, Veerle, Virginie en Yun. Het Labo Microbiologie wil ik danken voor hun gastvrijheid; als vreemde eend in de bijt kwam ik af en toe op bezoek met vragen over microbiologie of wat je van ons systeem vond, en vaak ook gewoon omdat het plezant was om langs te komen. In het bijzonder wil ik Wim De Smet danken voor de goede samenwerking die we gehad hebben en ik wens hem alle succes met het schrijven van zijn eigen doctoraat. Wim Gillis voor zijn ondersteuning in het ontwikkelen en uitbouwen van StrainInfo en het beheer van onze systemen.

I had the chance to perform this research in an international context, and meet incredible people during study leaves abroad. I still have good memories of the dynamic people I met during my short but intensive internship at the Knowledge Media institute (KMi) in Milton Keynes, UK. During my stay in the United States, I had the opportunity to live with great roommates: thank you, Nazgol, Nick, Pelle, Mia (who visited us for a month) and Sara. I also want to thank Gavin Tate (also known as 'Big D') for the exciting times and strange visitors he brought to the apartment. Terug in België vond ik schitterende huisgenoten in Joke en Celine. Ik wil ze bedanken voor het in huis halen van een doctoraatsstudent en alle grillen die daarbij horen. Ik beloof dat ik vanaf nu op tijd ga poetsen.

Voilà, we zijn er. Dit is een afscheid van een periode en een vertrouwde omgeving. Het is met spijt in het hart dat ik afscheid neem. Tegelijkertijd is dit ook een nieuwe start, en ik kijk al uit naar de nieuwe mogelijkheden en uitdagingen die voor me liggen. Ik wil iedereen bedanken voor de bevoorrechte kansen die ik gekregen heb. Ik wil ook jou, de lezer, bedanken voor het lezen van dit dankwoord. Ik hoop dat je volhoudt en ook de komende hoofstukken leest. Groeten!

Bert Verslyppe
Juli 2012

1 Introduction

They are almost everywhere you look, but you can not easily see them. Microorganisms form an essential component of life on earth [1, 2]. Although there were some presumptions, their discovery and understanding coincides with the development of microscopes. Antonie van Leeuwenhoek (born 1632), a tradesman and scientist from Delft (the Netherlands) built one of the first microscopes (see Figure 1.1). Using the microscopes he built himself, he was (among) the first to discover microorganisms. He discovered protists and bacteria which he called 'animalcules' in 1674 and 1676 respectively. In addition, he also discovered the cell vacuole, spermatozoa and the banded pattern of muscular fibers. I had a chance to see a replica of van Leeuwenhoek's microscope in the Museum of the History of Science in Oxford[1] (United Kingdom). Compared to the microscopes we have today, it is truly fascinating that such seemingly tiny instrument lead to fundamental biological discoveries.

The term microorganism is a collective name for many kinds of organisms. Following a definition given by the *Brock Biology of Microorganisms* – one of the main reference text books in the field of microbiology – a microorganism is a microscopic

[1]http://www.mhs.ox.ac.uk/

Figure 1.1: Antonie van Leeuwenhoek. Image taken from his book 'Den Waaragtigen Omloop des Bloeds' (1686) which is available in the public domain.

organism that comprises either a single cell (unicellular), cell clusters, or no cell at all (acellular) [3]. Microorganisms include bacteria, fungi, archaea, protists, microscopic plants (green algae) and animals such as plankton and planaria. As I had the opportunity to do my research in close collaboration with the Laboratory of Microbiology (Department of Biochemistry and Microbiology, Ghent University), a research group focusing on bacteria, my personal look into the microbiological world is especially focused on bacteria. Nevertheless, many principles of bacteria also apply to or easily translate to other kinds of microorganisms.

Microorganisms live at a micro (10^{-6}) scale. Bacterial cells for example, are typically $0.5 - 5.0$ µm in length, although exceptions exist. This affects many aspects of microbiological research, as investigating microbes requires specialized technology. Therefore, microbiology has been shaped by technological advances which allowed to investigate the specimens in new ways. For example, the development of genomic techniques lead to a different way of looking at an organism (genotype vs. phenotype). This lead to fundamental new insights such as Woese's three-domain phylogeny of cellular life which is based on the 16S rRNA genes of the cells [4]. As a computer scientist, I advocate that computer science will help to facilitate the next 'order of magnitude' shift in microbiology. Interestingly, this is made possible by recent advances in computer science itself. In my opinion, it started more or less twenty years ago with the advent of the Internet, and more specifically, the World Wide Web [5, 6].

In what follows, I will discuss some of the important microbiological concepts necessary to understand the remainder of this dissertation. I will explain the microbiology itself, as well as pointing to important differences between microbiology and computer science.

1.1 The microbiological workflow: a computer scientist's perspective

Overly simplified, microbiology could be described as the following series of steps: sampling, isolation, cultivation, description and publication. This simplification is unfair because microbiology is more than that; it is a wide domain with many different research directions, all with their specific know-how, rationale and customs. However, as a computer scientist, I seek to impose a model that organises what happens in microbiology. Essentially, this is driven by the need to understand which and how steps depend on each other, to be able to successfully build electronic systems for them. This model, based on general microbiology as I witnessed it during my research, is given in Figure 1.2 and was used throughout my research. However, from a microbiologist's perspective, my personal experience is that the true model probably is a lot more complex and less rigidly structured. Therefore, the model should be taken as a bird's eye view and considered with the necessary precaution.

1.1.1 Sampling

The work of a microbiologist starts with *sampling*, the act of *collecting* an *environmental sample*. The nature of this sample can be very diverse: an extract from soil, plants (e.g. the root of a tree, leaves,...), aquatic environments (oceans, seas, rivers, ponds,...), the human body (e.g. blood, tissue, wounds,...), and so on. Interestingly, a lot of microbiological diversity can be found in soil: one gram of typical soil contains populations ranging from 100 million to 3 billion bacteria. However, microbiologists are also often interested in so-called extreme environments: hot springs and volcanic areas, deep sea environments, polar environments, and so on.

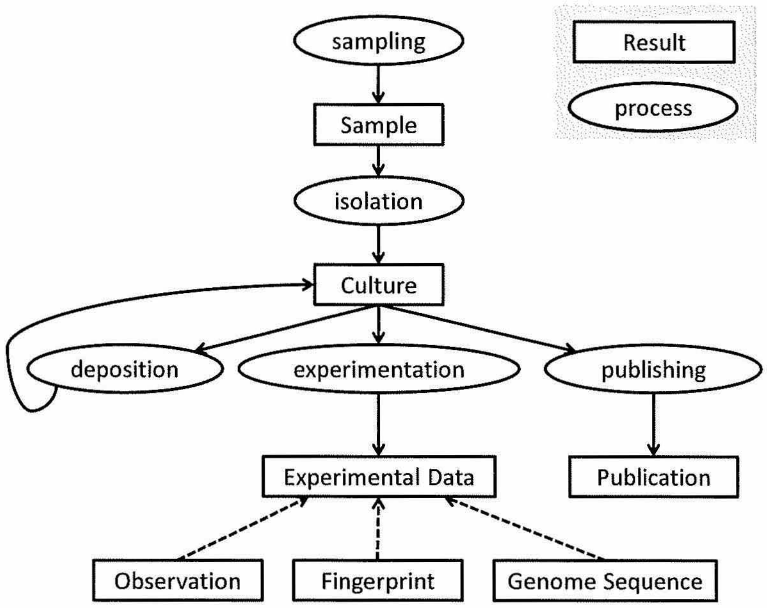


Figure 1.2: The microbiological workflow. This is a simplified model of typical microbiological research. Square rectangles denote a physical result, while ellipses correspond to processes.

Especially in those environments, challenging expeditions can be required to have access to the sampling sites.

This is one of the main differences between microbiology and computer science: biological research works with actual biological material! While computer scientists only instruct their computer what it needs to do, biologists also need to physically perform the work themselves. The nature of collecting one hundred articles about the summits of mountains from the Web is totally different from physically collecting a sample on those mountains. Therefore, computer scientists spend their time mostly on reasoning about the algorithmic aspects of a task, while biologists spend their time on executing their experiments. This difference also explains why biologists are willing (or in the worst case, have no other option) to do more tedious, repetitive tasks, for a longer time. In contrast, a colleague computer scientist once explained that if he needs to perform a task three or more times, he would write a script for it.

During the sampling process, it is important to consider the scale of microorganisms. Compared to their size, even a couple of centimetres is already a large distance. Therefore, GPS (Global Positioning System) coordinates of the sampling location need to be interpreted with care. Furthermore, a few millimetres could already give a big difference in the kind of environment the microorganism is living in. This is called the *habitat*, or as defined by a large online, crowd-sourced encyclopedia[2]:

> A *habitat* (which is Latin for 'in inhabits') is an ecological or environmental area that is inhabited by a particular species of animal, plant or other type of organism. It is the natural environment in which an organism lives, or the physical environment that surrounds (influences and is utilized by) a species population.

The habitat of a microorganism can for example be soil, the root of a tree, the human gut, and so on. Again, the scale of microorganisms affects the suitability of describing the habitat from which the sample was taken. In addition, determining the habitat is to some extent also a philosophical question [7]. Fish live in water, but is their habitat the water surrounding the fish, or is it the void that they have created in the water?

1.1.2 Isolation and growth

Isolation is the process that 'funnels' the environmental sample into an actual microorganism. Environmental samples often contain many different types of microorganisms and therefore isolation can be seen as a selection process that filters out one organism. It is performed by applying the sample on sterile growth medium. This is often a petri dish, filled with a layer of agar (agarose gel). Once the growth medium is inoculated, the plates are incubated at the optimal temperature for growing the selected bacteria. This means that the bacteria are given some time at a temperature and oxygen condition they prefer. In general, the temperature depends on the environment or habitat the sample was extracted from: for example, bacteria taken from the human body often proliferate at 37 °C, while bacteria taken from soil need lower temperatures. It is also possible to grow the microorganisms in a liquid growth medium, called a liquid broth.

[2]See `http://en.wikipedia.org/wiki/Habitat`

Bacteria grow up to a fixed size and then reproduce through binary fission, a form of asexual reproduction. By cell division, two identical clone cells are produced out of the original ancestor cell. This is an exponential process that doubles the number of microorganisms in each generation, until the agar plate is fully colonised or the nutrients are depleted. The duration of a generation depends of the kind of microorganism and the conditions. In laboratory conditions, this can be as short as ten minutes [8], but for many species ranges between twenty and sixty minutes [9]. The end result is a *colony* of cells with identical genomic material. This is a central concept in microbiology: growing bacteria yields copies of exactly the same bacteria. The copies have (virtually) the same genomic material, and therefore the same properties. This makes them interesting material for research – and is an essential feature for the research performed in this dissertation – as almost an infinite supply of equivalent microbiological material can be generated starting from one single bacterial cell.

This still does not explain how isolation is performed. When bacteria are picked up with a sterile metal rod and transferred to another plate, they can be diluted by employing a special way of streaking bacteria on the agar plate (see Figure 1.3). Due to the scale of microorganisms, many cells are picked up and transferred to the new plate in this operation. However, the streaking pattern results in a distribution of varying colony densities on the plate. After incubation, there can be growth of different colonies with different colours and shapes on the resulting agar plate if the inoculum contains different kinds of bacteria. In this case, a colony is chosen, picked up, transferred to a new plate and incubated. The isolation process (picking and streaking) is repeated until an agar plate with uniform colonies is obtained. At this point, the bacteria are said to be in *pure culture*, which means that all cells track back to the same bacterial cell and that therefore no other kinds of bacteria are growing.

1.1.3 Cultures and strains

Once in pure culture, a stage of maintaining the microorganism is started. As long as some cells of the microorganism are transferred to a new agar plate before the nutrients on the current plate are exhausted, the microorganism can be maintained *in vitro*. Due to the nature of binary fission, which generates microbial offspring with identical genetic material, the microorganisms on these plates are considered equivalent. Evidently, it is also possible to create multiple derivative plates from a single original plate containing the pure culture. This allows different experiments

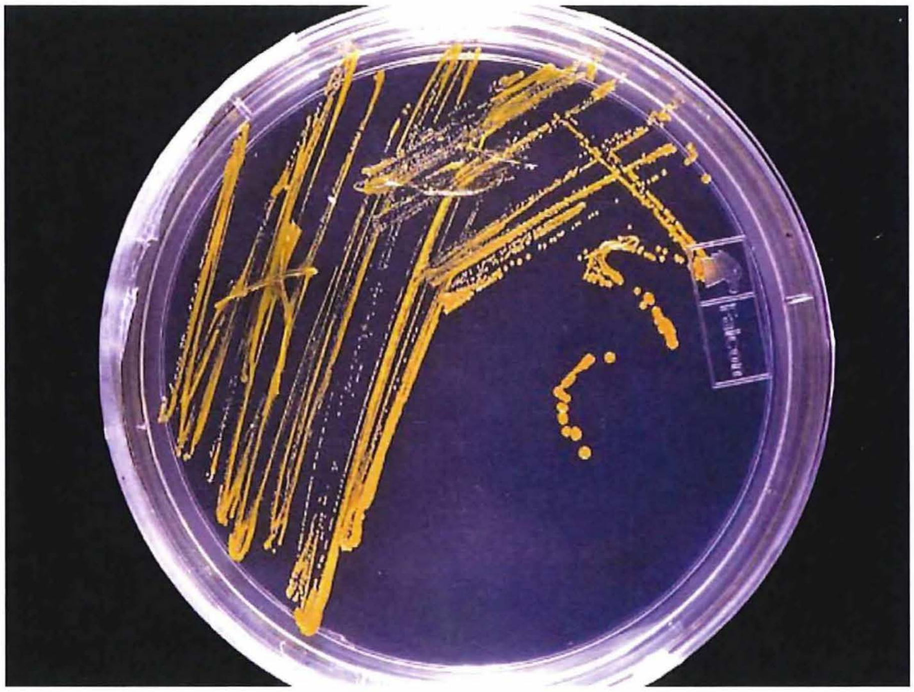


Figure 1.3: Petri dish with agar containing the typical scratching pattern used during isolation. Image taken from http://oceanexplorer.noaa.gov/explorations/03bio/background/microbiology/microbiology.html (public domain).

to be performed on independent instances of the source material, or the material can be sent to other researchers, creating a distribution of the material. As a result, this creates an abstract notion of the ensemble of all equivalent cultures. In computer science terminology, this notion is referred to as the *class* to which *instances* belong. The class of equivalent microorganisms is important to consider, as this links independent experiments and observations together to the same organism. Microbiology uses the term *strain* for this.

For what follows, it is important to define the *strain* and *culture* concepts. According to the definition of Staley and Krieg [10], the term *culture* is used to indicate a population of cells of a strain, instantiated at a given place during a given time, e.g. in a test tube, on an agar plate or in a cryopreserved or lyophilized state intended for long-term preservation. The term *strain* is used to refer to the assembly of all descendants of a single isolation in pure culture, usually a succession of cultures ultimately derived from a single colony. Although more theoretical interpretations of the strain and culture concepts exist [11], we will stick to the above definitions as they can be closely followed in software models and reflect current practice in bacterial nomenclature [12].

Put simply, a strain is a collection of identical cultures. However, in practice, there can be a small factor of genetic change (i.e. random errors that occur during duplication of the genetic material), which could result in a small variability. Therefore, we use the term *equivalent* to denote that in practice the cultures can be used interchangeably. In addition, maintaining cultures imposes risks of contamination, which means that cultures that actually belong to another strain become mixed in or even replace the original strain. This explains why the strain concept in practice is less *crisp* or ascertainable compared to similar concepts in computer science or mathematics.

1.1.4 Biological Resource Centers (BRCs)

An important aspect of scientific provenance in microbiological research is the collection of microbial material. Therefore, after isolation, identification and description, bacterial and archaeal strains are typically deposited in internationally recognised Biological Resource Centers (BRCs), previously called 'culture collections' [13–15]. The OECD (Organisation for Economic Co-operation and Development) defines best practice guidelines which culture collections need to implement to conform to the BRC concept [16,17]. BRCs perform long-term preservation of microorganisms, establish quality control procedures to safeguard authenticity and provide worldwide distribution to researchers and companies according to international legislation on the dissemination of biological material. This allows scientific progress as it allows to repeat and build upon previous work. The cultures themselves are preserved long term using special techniques (e.g. lyophilization and cryopreservation) making it possible to revive them at a later stage. The term BRC is used throughout this dissertation as it is more restrictive than the term culture collection by suggesting that a quality management system in line with OECD guidelines is in place and it is also defined for collections of other types of material. However, as no official certification procedure is implemented, it is not possible to become officially recognised as a BRC endorsed by the OECD.

Upon receipt of new biological material, BRCs issue identifiers (so-called 'strain

Strain number	BRC (Country)
ATCC 14579	American Type Culture Collection (U.S.A.)
CIP 66.24	Collection de L'Institut Pasteur (France)
DSM 31	Deutsche Sammlung von Mikroorganismen und Zellkulturen GmbH (Germany)
LMG 6923	Belgian Coordinated Collections of Microorganisms / LMG Bacteria Collection (Belgium)
VKM B-504	All-Russian Collection of Microorganisms (Russian Federation)

Table 1.1: Example strain numbers and the BRCs they belong to. All strain numbers in this example are equivalent, or in other words, denote the same strain.

numbers') when depositing the material in their holdings. *Strain numbers* are used to refer to the biological material in publications or to retrieve material from BRCs for further analysis or scientific validation. Strain numbers are usually composed of a BRC acronym (possibly pointing to a specific subcollection) followed by a number that uniquely identifies the culture in the BRC. Many variations on this general labeling scheme have been adopted. Table 1.1 contains some example strain numbers.

A large fraction of the biological material entering BRCs is made available for further distribution. These public holdings are listed in *catalogs* that were traditionally published in print, but are at present often accessible online. Catalogs provide supplementary information about microorganisms, ranging from administrative information such as isolation and deposit details to richer information such as taxonomic identifications, biochemical and phenotypic properties, genomic traits and related literature. Strain numbers are locally unique identifiers that researchers and other parties can use to order particular cultures from a BRC.

BRCs exchange microbial material, effectively distributing the material worldwide. Each time material gets deposited, a new strain number is issued by the BRC to facilitate its accession. The process of deposit and assignment of new strain numbers results in an exchange history of the corresponding strain, in which each culture is represented by its corresponding strain number. Strain numbers that trace back to the same isolate can be used interchangeably, because they represent subcultures of the same biological material. Such strain numbers are considered to be equivalent. Therefore, in order to find all information available on a given strain, all equivalent strain numbers should be used as a collective search key.

Most information resources will typically list only a few but not all equivalent strain numbers, so that in practice, not all existing information can be easily found and scientific experiments (such as genome sequences) are often redundantly performed due to ignorance of equivalent results.

1.1.5 Taxonomy: identification and classification

The domain of taxonomy encompasses classification, nomenclature and identification. *Classification* is the orderly arrangement of organisms into taxonomic groups (taxa) on the basis of similarity. *Nomenclature* refers to the labelling of the defined taxonomic units (i.e. giving a proper name). *Identification* is the process of determining that a new isolate in pure culture belongs to one of the established named taxa. *Typing* is related to identification: it is the process of recognising and classifying isolates as members of the same or different strains.

Genetic drift and evolution (physical separation and periods of selection) lead to variation in bacterial genomes, changing their size, organisation, content and sequence. The extent to which genetic drift occurs depends on the species (some species are extremely stable while others are variable) and the environment (some environments might pose a stronger stress on strains). Genomes of species are composed of a core set of genes that are conserved among strains of the same species (i.e. the *core genome*) and accessory genes that are strain specific. The content and size of the core genome varies with the species. However, there is a practical need to define bacterial species as a name bears information about its properties. Related to this is the question whether delineated (or mathematically speaking, discrete) bacterial species exist, given the genomic evolution.

There is a need to define the concept of a (bacterial) species, but it is hard to come up with a conclusive definition. Breed *et al.* state: 'The unit of classification is a coherent group of like individuals, called a species. The term is difficult to define with precision because a species is not a definite entity, but a taxonomic concept' [18]. Staley and Krieg define a *species* as a collection of strains that share many features in common and differ considerably from other strains [10]. Groups of species form genera, which on their turn form families, orders, classes, phyla, kingdoms and up to the highest-level taxon, domains. In binomial nomenclature, species names consist of a genus name and a species epithet. For example, *Escherichia coli* is the species name of a well-known microorganism commonly found in the intestines of warm-blooded organisms. If the genus name can be unambiguously determined from the context, it is often abbreviated to its first letter (e.g. *E. coli*).

Progress in the field of taxonomy has been dominated by technological progress. Conventional bacterial taxonomy puts heavy emphasis on the analysis of phenotypic properties of the organism. In the 1950s and 1960s (the advent of genomic techniques and computers), it became evident that the analysis from a large number of perspectives provided a more stable classification and a superior means to classify and identify bacteria. This led to *polyphasic taxonomy*: a consensus approach to bacterial taxonomy and the species definition which integrates several generally accepted techniques for the classification of bacteria [19]. The polyphasic species is an assemblage of isolates originating from a common ancestor population in which a steady generation of genetic diversity resulted in cultures with different degrees of recombination and is characterised by a certain degree of phenotypic consistency, a significant degree of DNA-DNA hybridisation (70%, although this number is somewhat arbitrarily established) and over 97% of 16S rRNA gene sequence similarity. Although there is a growing availability of genomic techniques, it remains unclear how a genomic species concept can be fitted [20].

Each species is appointed a type strain. This strain is arbitrarily chosen (often being the first isolate) and is taken as a name bearer for the species. Scientific recognition of new (prokaryotic) species encompasses deposition of a type strain and the official publication of the new species name and its formal description. A viable culture of a type strain must be deposited in at least two publicly accessible service collections in different countries from which cultures must be available [21]. The importance of type strains from a taxonomic perspective is often an incentive for their distribution. As a consequence, type strains have generally speaking a broader international distribution and therefore more equivalent cultures as regular strains. Furthermore, strain numbers of type strains are appended with a superscript `T` to indicate their type strain status (e.g. `LMG 6923`^{T}). This can be somewhat misleading as the type strain status is not bound to the strain itself, but to the species for which the strain is the type strain. It can also be the case that a strain is the type strain of two or more species (the type strain stays bound to the species definition, even if species are redefined).

1.1.6 Empirical data

Although the nature of experiments and their results is very diverse, there are some broad categories that can be distinguished. Phenotypic observations can be made through the use of biochemical techniques and optical instruments. Often staining methods are employed to enhance contrast or to make certain structures visible.

Fingerprints in the form of scattered bands patterns are generated by denaturing the cells and using gel electrophoresis to generate a spread of molecules. There are many different techniques that aim at different molecules in the cell, but often DNA (which is amplified using polymerase chain reaction) is used. Other techniques use different ways of measuring the occurrence and complexity of molecules in the cell (such as MALDI-TOF and Raman spectroscopy), and the results can be represented as a two dimensional graph. Furthermore, genome sequencing allows to obtain the full blueprint of an organism, expressed as string consisting of the letters 'ACGT' (DNA) or 'ACGU' (RNA). Up to date, complete genome sequences are the most complete way of assessing the genotype of an organism.

1.1.7 Capturing science: publication of findings

It is common practice to capture scientific results and insights in literature, and this is a central aspect of scientific culture. Science progresses through iteratively describing findings in research papers, letters or reports and by publishing them to the scientific community. Publications capture findings and preserve them for the future. Each publication therefore can be seen as a checkpoint of the state of the art Even if publications become outdated or superseded by new findings, the historical legacy of scientific literature remains important to understand the context in which scientific progress was made.

Although about one century ago publications that systematically describe microorganisms already appeared, this is still done nowadays – albeit at a larger scale and with higher precision and sophistication. The constantly improving technology has changed the interpretation and precision of the work, but the essence of microbiological research still remains the same: discovery, description and classification of microbiological material.

The *International Journal of Systematic and Evolutionary Microbiology* (IJSEM) is regarded as the *de facto* standard journal for publications that contain descriptions of new species. The Bacteriological Code [22] contains the principles to which those descriptions must conform to validly describe new species. Descriptions can be published in the IJSEM or other journals. The *List of Prokaryotic names with Standing in Nomenclature*[3] maintained by J.P. Euzéby keeps track of all validly published species.

[3]`http://www.bacterio.cict.fr/`

Part I

The StrainInfo platform

2 The StrainInfo platform

The continuous expansion of microbiological research has produced a vast body of knowledge. Next to information provided in BRC catalogs, morphologic, biochemical, genomic, taxonomic and ecological knowledge got dispersed over journals, sequence repositories, taxonomic databases and other resources. As increasing amounts of data become electronically available, we witness an increasing research interest towards in-depth analysis of data accumulated over time instead of generating new data as sole output of experimental work. Such accumulative research requires electronic access to standardized data in public databases. However, most bioinformatics databases collect experimental results and somewhat neglect their common ground: the actual microorganisms on which experiments have been performed. As important as it is to keep track of the original biological material, we also need to properly document related meta-information. Microbiological research has a long tradition of preserving microbial diversity, which until now was merely focused on physical storage and distribution of microorganisms. Meta-information maintained by individual BRCs is distributed over many online catalogs, each using their own interface and data formats. Large-scale organism-centric analysis, however, requires an integrated view of this information [23].

This chapter is outlined as follows. Section 2.1 discusses the concepts of passports and browsers as introduced by StrainInfo. This includes strain passports and browsers (Sections 2.1.1 and 2.1.2) and similarly organized taxon, sequence and publication passports (Section 2.1.3). Section 2.2.1 further elaborates on the integration process used to construct passports and browsers. Section 2.2.2 describes how the passport and browser concepts relate to other integration initiatives. StrainInfo also introduces culture URIs: globally unique and dereferenceable identifiers associated with cultures that allow electronic access to strain passports. This URI scheme is described in detail in Sections 2.2.4 and 2.2.5. Technical implementation details are discussed in Chapter 3.

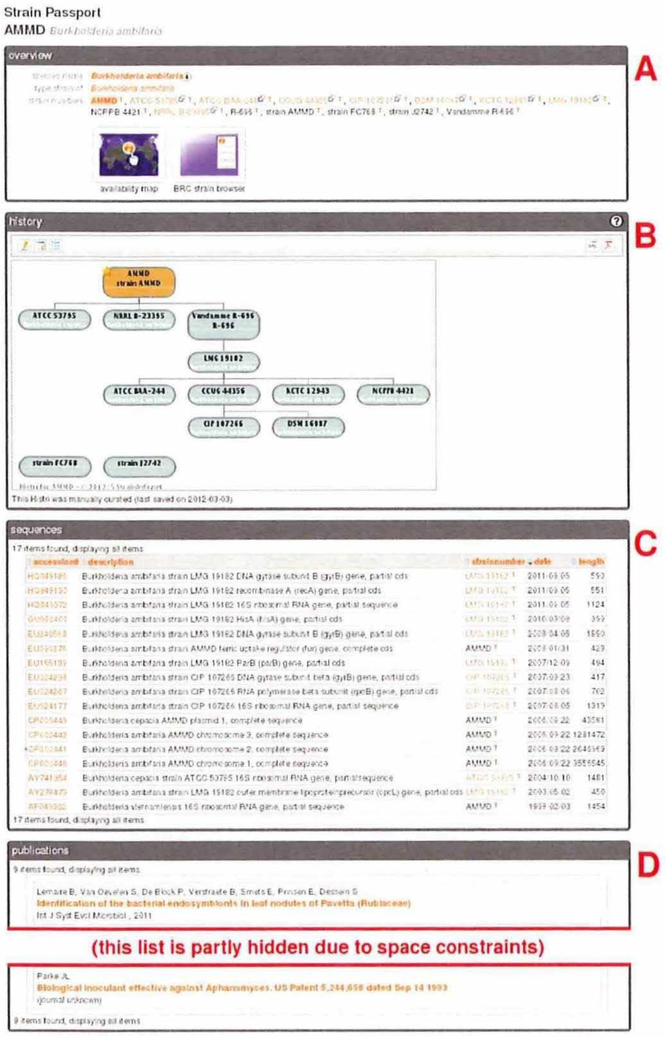
2.1 Integrated access to microbial information

StrainInfo[1] is a software platform designed around the concept of passports and browsers, with the aim to provide and display integrated information about microorganisms. Each passport gives an integrated view on electronically accessible data collected for a particular type of object. This bundles the retrieved information into a single point of access. Each type of passport also has an associated browser that offers direct access to the primary data that underpins the integration process of the passports. This enables easy navigation to more detailed information that was left out of the integration process and allows validation of information provided by passports. The ideas behind both concepts are illustrated by starting our discussion for the particular case of strain passports and strain browsers.

2.1.1 Strain passports summarize information on microorganisms

Strain passports are organism-centric and intend to integrate everything StrainInfo knows about a particular microorganism at the strain level. Figure 2.1 displays the typical outline of a strain passport, using the *Burkholderia ambifaria* type strain [24] as an example. Passports are subdivided into separate sections that describe different aspects of the object they represent. Each passport starts with an overview section containing general information that identifies the object and describes its primary features. The overview section of strain passports, for example, lists all equivalent strain numbers known to be assigned to the strain and all taxa for which it is the type strain (see Figure 2.1.A). It also lists all taxonomic names under which

[1]`http://www.straininfo.net`



Strain Passport

AMMD *Burkholderia ambifaria*

A

overview

species name	*Burkholderia ambifaria*
type strain of	*Burkholderia ambifaria*
strain numbers	AMMD T, ATCC 53795 T, ATCC BAA-244 T, CCUG 44356 T, CIP 107266 T, DSM 16087 T, KCTC 12943 T, LMG 19182 T, NCPPB 4421 T, NRRL B-23395 T, R-696 T, strain AMMD T, strain FC768 T, strain J2742 T, Vandamme R-696 T

B

history

This Histo was manually curated (last saved on 2012-03-03)

C

sequences

17 items found, displaying all items

accession#	description	strainnumber	date	length
HQ849185	Burkholderia ambifaria strain LMG 19182 DNA gyrase subunit B (gyrB) gene, partial cds	LMG 19182 T	2011-09-05	593
HQ849152	Burkholderia ambifaria strain LMG 19182 recombinase A (recA) gene, partial cds	LMG 19182 T	2011-09-05	551
HQ841572	Burkholderia ambifaria strain LMG 19182 16S ribosomal RNA gene, partial sequence	LMG 19182 T	2011-09-05	1124
[illegible]	Burkholderia ambifaria strain LMG 19182 HisA (hisA) gene, partial cds	LMG 19182 T	2010-03-08	393
[illegible]	Burkholderia ambifaria strain LMG 19182 DNA gyrase subunit B (gyrB) gene, partial cds	LMG 19182 T	2008-04-05	1650
[illegible]	Burkholderia ambifaria strain AMMD ferric uptake regulator (fur) gene, complete cds	AMMD T	2008-01-31	429
EU165197	Burkholderia ambifaria strain LMG 19182 ParB (parB) gene, partial cds	LMG 19182 T	2007-12-09	494
EU024298	Burkholderia ambifaria strain CIP 107266 DNA gyrase subunit beta (gyrB) gene, partial cds	CIP 107266 T	2007-09-23	417
EU024267	Burkholderia ambifaria strain CIP 107266 RNA polymerase beta subunit (rpoB) gene, partial cds	CIP 107266 T	2007-08-06	702
EU024177	Burkholderia ambifaria strain CIP 107266 16S ribosomal RNA gene, partial sequence	CIP 107266 T	2007-08-05	1313
CP000443	Burkholderia cepacia AMMD plasmid 1, complete sequence	AMMD T	2006-09-22	43581
CP000442	Burkholderia ambifaria AMMD chromosome 3, complete sequence	AMMD T	2006-09-22	1281472
CP000441	Burkholderia ambifaria AMMD chromosome 2, complete sequence	AMMD T	2006-09-22	2646969
CP000440	Burkholderia ambifaria AMMD chromosome 1, complete sequence	AMMD T	2006-09-22	3556545
AY741354	Burkholderia cepacia strain ATCC 53795 16S ribosomal RNA gene, partial sequence	ATCC 53795 T	2004-10-10	1481
AY279475	Burkholderia ambifaria strain LMG 19182 outer membrane lipoprotein precursor (opcL) gene, partial cds	LMG 19182 T	2003-05-02	450
[illegible]	Burkholderia vietnamiensis 16S ribosomal RNA gene, partial sequence	AMMD T	1999-02-03	1454

17 items found, displaying all items

D

publications

9 items found, displaying all items

Lemaire B, Van Oevelen S, De Block P, Verstraete B, Smets E, Prinsen E, Dessein S
Identification of the bacterial endosymbionts in leaf nodules of Pavetta (Rubiaceae)
Int J Syst Evol Microbiol , 2011

(this list is partly hidden due to space constraints)

Parke JL
Biological inoculant effective against Aphanomyces. US Patent 5,244,658 dated Sep 14 1993
(journal unknown)

9 items found, displaying all items

Figure 2.1: Strain passport of the *Burkholderia ambifaria* type strain, showing separate sections with overview information (A), exhange history (B), genome sequences (C) and references to the published literature (D) for this particular strain. This strain passport can be found at `http://www.straininfo.net/strains/355609`.

the strain is identified by the different BRCs, possibly showing differences in opinion, and displays a concensus name resulting from a majority vote selection procedure. Strain numbers appearing on strain passports always link into the corresponding strain browser and taxonomic names appearing on passports always link into the corresponding taxon passport (described further on). BRCs that have one or more cultures of the strain in their holdings can be geographically located on a map. This information serves as a practical tool for selecting the most suited BRC for ordering the required biological material (e.g. in function of shipment or import/export regulations).

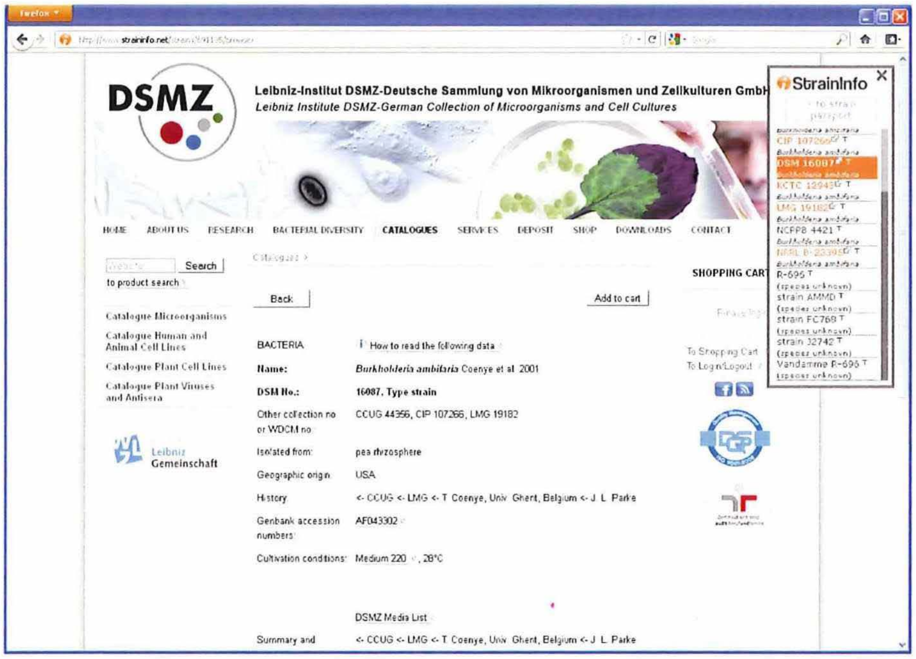
Microbial reference strains are often available from multiple BRCs as a result of multiple deposits or strain transfers between collections and researchers. Upon receipt of a culture into a BRC, a new strain number that is equivalent to all other strain numbers previously assigned to the strain is issued. Equivalent strain numbers can be used interchangeably as they all correspond to cultures of the same biological material at another place or another point in time. To keep track of the relationship between equivalent strain numbers, the history section of a strain passport depicts the completely integrated exchange history of the strain, using a representation called a 'Histri' (see Figure 2.1.B). Successive assignment of strain numbers is visualised as rooted trees that start from the original isolate and branch down after each culture transfer. This helps to understand and verify the equivalence of strain numbers. Histris result from a semi-automated reconstruction process that draws upon historical information extracted from BRC catalogs and other resources (see Chapter 4). Strain passports also link to an editor for manual Histri updates to take advantage of community curation.

The sequence section of a strain passport lists all International Nucleotide Sequence Database Collaboration (INSDC) [25] records that are sequenced from a culture of the corresponding strain (see Figure 2.1.C). Relevant sequences are found by indexing particular information fields from the European Molecular Biology Laboratory (EMBL) Nucleotide Sequence Database records and resolving ambiguous strain number references by taking into account contextual data. It is important to note here that knowledge about all equivalent strain numbers can be used to retrieve an exhaustive list of sequences derived from any culture of the strain. The importance of having integrated strain number information for sequence retrieval is illustrated by the example strain in Figure 2.1. It shows that seventeen sequence records were found for the *B. ambifaria* type strain at the time of writing. Seven of them were deposited into the INDSC database with a reference to strain number `LMG 19182`, three with a reference to strain number `CIP 107266` and one with a reference to strain number ATCC 53795. In addition, six sequences were deposited with a reference to strain number AMMD, an identifier assigned in 1985 to the original isolate of the *B. ambifaria* type strain by J.L. Parke of the Department of Crop and Soil Sciences, Oregon State University, United States. Note that this strain only became the type strain of *B. ambifaria* in 2001, 16 years after its isolation [24]. This explains why some BRC catalogs and sequence records still identify the strain as *B. cepacia*, its taxonomic name before it became a type strain of *B. ambifaria*. Using the general retrieval strategy for downstream information that was outlined above, strain passports can provide all sequences derived for a particular strain, irrespective of the strain number and taxonomic name used when sequences were deposited into the INSDC databases.

A similar retrieval strategy can be applied to find publications related to a particular strain. Results of such a strategy are shown in the literature section of strain passports where references to relevant publications are listed (see Figure 2.1.D). These references were either extracted directly from BRC catalogs or infered indirectly from INSDC sequence records linked to a given strain. For the time being, the literature index is limited to those reference sources. Search results could be extended by extracting strain number references from the abstract or full text of research papers and patent applications, thereby resolving ambiguities by taking into account some contextual information, but this has not been implemented yet. Knowledge about the equivalence relation between strain numbers is again used to bundle literature on a given strain irrespective of the references to its strain numbers. In practice, multiple resources might couple the same publication to the same strain, possibly using different equivalent strain numbers. An additional deduplication step is thus required to guarantee that literature references are listed only once. This is accomplished using an automated mapping procedure that resolves literature references to PubMed identifiers and Digital Object Identifiers (DOI).

2.1.2 Strain browsers provide deep links into BRC catalogs

Rather than replacing online catalogs of individual BRCs by strain passports, StrainInfo envisions to augment the autonomous and heterogeneous network these catalogs establish with a uniform and integrated access protocol. Therefore, strain numbers assigned by BRCs that have an online catalog will always appear on strain passports with a link into a strain browser. Clicking such a hyperlinked strain number causes the strain browser to locate and display the corresponding BRC catalog entry (Figure 2.2). Differences and changes in the access methods used for online


DSMZ
Leibniz-Institut DSMZ-Deutsche Sammlung von Mikroorganismen und Zellkulturen GmbH
Leibniz Institute DSMZ-German Collection of Microorganisms and Cell Cultures
HOME
ABOUT US
RESEARCH
BACTERIAL DIVERSITY
CATALOGUES
SERVICES
DEPOSIT
SHOP
DOWNLOADS
CONTACT
Search
to product search
Catalogue Microorganisms
Catalogue Human and Animal Cell Lines
Catalogue Plant Cell Lines
Catalogue Plant Viruses and Antisera
Leibniz Gemeinschaft
Back
Add to cart
SHOPPING CART
To Shopping Cart
To Login/Logout
StrainInfo

BACTERIA — How to read the following data

Name: *Burkholderia ambifaria* Coenye et al. 2001

DSM No.: **16087, Type strain**

Other collection no or WDCM no: CCUG 44356, CIP 107266, LMG 19182

Isolated from: pea rhizosphere

Geographic origin: USA

History: <- CCUG <- LMG <- T. Coenye, Univ. Ghent, Belgium <- J. L. Parke

Genbank accession numbers: AF043302

Cultivation conditions: Medium 220, 28°C

DSMZ Media List

Summary and: <- CCUG <- LMG <- T. Coenye, Univ. Ghent, Belgium <- J. L. Parke

CIP 107266 T — *Burkholderia ambifaria*
DSM 16087 T — *Burkholderia ambifaria*
KCTC 12943 T — *Burkholderia ambifaria*
LMG 19182 T — *Burkholderia ambifaria*
NCPPB 4421 T — *Burkholderia ambifaria*
NRRL B-23395 T — *Burkholderia ambifaria*
R-696 T — (species unknown)
strain AMMD T — (species unknown)
strain FC768 T — (species unknown)
strain J2742 T — (species unknown)
Vandamme R-696 T — (species unknown)

Figure 2.2: Strain browser displaying entry DSM 16087^T (*Burkholderia ambifaria*) in the online catalog of the Deutsche Sammlung von Mikroorganismen und Zellkulturen GmbH (DSMZ). DSM 16087^T is one of the equivalent strain numbers assigned to the *Burkholderia ambifaria* type strain. Using the overlay panel on the right, users can navigate to online catalog entries of other BRCs that have this strain in their holdings. This strain browser can be found at `http://www.straininfo.net/strains/691135/browser`.

catalogs of BRCs are abstracted away by strain browsers, that provide deep links directly to the catalog entries themselves, thus removing the burden from searching through different web interfaces.

In addition, strain browsers add an extra overlay panel that lists all equivalent strain numbers known to StrainInfo for the displayed catalog entry. Listed strain numbers that were issued by BRCs that have an online catalog will again cause the strain browser to locate and display the corresponding BRC catalog entry. As a result, strain browsers group individual catalog entries that provide information on the same microbial strain by bundling their hyperlinks into the overlay panel. This allows easy navigation between related catalog entries for comparison and cross-validation. BRC catalog entries can provide detailed information about microorganisms that has not been integrated into the strain passports yet, or simply falls outside the StrainInfo integration scope (e.g. information on ordering a culture of the strain from a particular BRC).

2.1.3 Beyond strain passports and strain browsers

Apart from strain passports and strain browsers, StrainInfo also applies both general integration concepts to other types of objects: taxa, sequences and literature references. The different kinds of passports thus provide hooks for displaying inverse relationships between different objects and for external data providers to link into StrainInfo.

Whereas strain passports identify individual strains to certain taxonomic names, one might also want to find all strains that were identified to a certain taxon. Taxon passports are used to consolidate these relationships between strains and taxa: taxonomic names appearing on passports always link into the corresponding taxon passport and taxon passports contain a feature to search for strains that were assigned to the taxon, irrespective of the BRC into which these strains were deposited. StrainInfo aggregates taxonomic information for its taxon passports from several external resources: the List of Prokaryotic names with Standing in Nomenclature[2] (LPSN) [26] maintained by Dr. Jean Euzéby, the Bacterial Nomenclature Up-to-Date[3] published by the Deutsche Sammlung von Mikroorganismen und Zellkulturen GmbH (DSMZ) and Mycobank[4] [27] maintained by the Centraalbureau voor Schimmelcultures (CBS-KNAW) Fungal Biodiversity Centre. Taxon passports

[2]http://www.bacterio.cict.fr/
[3]http://www.dsmz.de/bacterial-diversity/bacterial-nomenclature-up-to-date.html
[4]http://www.mycobank.org/

are hierarchically structured, with downward and upward links to subtaxa and the parent taxon. If a taxon has an appointed type strain (which is the case if the taxon is a species), this information is also given on its taxon passport with a link into the corresponding strain passport. Taxon passports have an associated taxon browser that allows navigating primary resources that provide more detailed taxonomic information: LPSN, Bacterial Nomenclature Up-to-Date, Mycobank, NCBI taxonomy[5] [28] and WikiSpecies[6]. Analogous to strain browsers, taxon browsers provide an overlay panel which bundles the resources that serve information on the same taxon.

Each INSDC sequence record that relates to information in StrainInfo has an associated sequence passport. Sequence passports display meta-information about their corresponding sequence, including a link into the strain passport of the organism from which the sequence was derived. This allows users to find more information on the sequenced organism beyond what is provided by the INSDC. As discussed above, strain passports display the reverse relationship by listing those sequence passports that represent sequences derived from the cultures of the corresponding strain. Sequence passports have an associated sequence browser with an overlay panel bundling resources that serve detailed information on the corresponding sequence record. These, of course, include the three databases that take part in the INSDC: GenBank [29], the European Nucleotide Archive (ENA) [30] and the DNA data bank of Japan (DDBJ) [31]. Additional links are provided into the small and large subunit rRNA databases of SILVA [32]. Sequence passports may thus be seen as localized versions of the NCBI LinkOut system [33] with a specific role to establish persistent links between cultured organisms deposited into BRCs and related sequence records in the INSDC databases.

Literature references made on strain, taxon and sequence passports all have their own publication passport. These publication passports contain detailed citation information in their overview section and provide links to electronic versions of the publication. As expected, these links are displayed by an associated publication browser. Publication passports also list all strains and sequences that appear in the publication.

[5]http://www.ncbi.nlm.nih.gov/taxonomy
[6]http://species.wikimedia.org

2.1.4 Locating passports

StrainInfo has two built-in features for locating passports. The simplest function – called Smart Search – infers from given search terms what kind of objects the user is looking for: strains, taxa or sequences. It uses a heuristic approach to discriminate strain numbers from taxonomic names and sequence accession numbers. Search terms for Smart Search can be entered from all StrainInfo web pages, which is the preferred way for users to search StrainInfo. Smart Search automatically redirects to passport pages if search terms identify an object in a unique way. If there are multiple hits, the corresponding passports are listed on a search results page. In addition to Smart Search, StrainInfo also implements a more classic advanced search function. It allows to search for strains based on a complex query composed of (partial) strain numbers, taxonomic names, type strain status, sequence accession numbers and full text searches through BRC catalogs.

Apart from built-in search features, it is worth noting that about 44% of the users enter passports using their favorite search engines or after following direct links to StrainInfo offered by external data providers. NCBI LinkOut was used, for example, to link GenBank and Genome Project records to strain passports, NCBI taxonomy records to taxon passports and PubMed records to publication records. ENA, SILVA, LPSN and a growing number of online BRC catalogs also provide links to strain passports. All these links make use of a stable URI scheme provided by StrainInfo, which is discussed later on.

2.1.5 Web services

Where StrainInfo as a web application provides integrated microbial information in human readable form, large-scale bioinformatics projects need to access the same underlying integration results in machine readable form [34]. Therefore, StrainInfo provides various web services, as many other bioinformatics resources do. Web services are online programming interfaces that offer machine-to-machine access to data or algorithms. They consume and produce structured data in machine-processable format. As such, web services from different providers can easily be composed into complex data processing pipelines, an approach exploited by workflow management systems such as the Taverna Workbench [35, 36].

StrainInfo implements RESTful web services (Representational State Transfer) that provide machine-readable representations of its passports and additional SOAP (Simple Object Access Protocol) web services for advanced queries. RESTful web services require less programming overhead to collect information, and are therefore preferred over SOAP web services. Using the RESTful approach, strain passports are available in XML (Extensible Markup Language) and RDF (Resource Description Framework) formats. In addition, custom exports that perform specialized queries on the StrainInfo index give access to information that cannot easily be retrieved by browsing the web interface. All exports are listed on an overview page[7]. Advanced queries and other computationally intensive procedures have been implemented as SOAP web services. They are described in more detail in StrainInfo's online help[8].

2.2 Discussion

Central to the general design of StrainInfo is the integration of the equivalent strain numbers that are assigned to different cultures of the same strain. As discussed in Section 1.1.3, we follow the definition of the terms strain and culture as given by Staley and Krieg [10]. They define a *strain* as all descendants of a single isolation in pure culture, usually a succession of cultures ultimately derived from a single colony. A *culture* is a population of (bacterial) cells, instantiated in a given place during a given time, e.g. on an agar plate.

It is important to understand that strains are abstract concepts, whereas cultures represent the physical biological material with which experiments and protocols are carried out. Strain numbers are used to label cultures. In that sense, the term strain number might actually be misleading and use of the term *culture number* would have been more appropriate. However, we will continue to speak about strain numbers, as this term is commonly used in the field. In practice, strain numbers are used to indicate both a particular culture of a strain (e.g. a culture obtained from the BRC whose acronym is part of the strain number) or the strain as a whole. In the latter case, strain number `LMG 19182`^{T} can be used as a placeholder to point to the type strain of *B. ambifaria* rather than pointing to the particular culture of this strain held in the BCCM^{TM}/LMG Bacteria Collection at Ghent University, Belgium. The context most often suggests implicitly whether strain numbers are used for referencing particular cultures or strains as a whole. Experiments are always done on cultures, but usually to determine general properties of the strain. A genome sequence, for example, is derived from a particular culture, but should in theory be the same for all cultures of the strain (not taking into account possible mutations

[7] `http://www.straininfo.net/exports`

[8] `http://www.straininfo.net/docs/webservices`

or contaminations). It is therefore common to say that the INSDC sequence record with accession number CP000440, which has been deposited with a reference to strain number AMMD^T, is the complete genome sequence of the first chromosome of the *B. ambifaria* type strain.

2.2.1 Integrating cultures into strains

StrainInfo merges equivalent strain numbers into individual strain objects using a special-purpose equivalence integration algorithm. In brief, the equivalence relation of strain numbers is computed as the transitive closure of pairwise equivalences encoded in the `Other Collection Numbers` fields extracted from BRC catalog entries. This computation is done using an incremental union-find algorithm to assure accumulative learning. In practice, the correctness of processing the equivalence of strain numbers is seriously hampered by the ambiguity of strain numbers that are assigned as locally unique identifiers without guarantee of their uniqueness in a global information environment and false equivalences introduced by human errors such as typos in BRC catalogs. This results in numerous false positive and false negative equivalence classes if no countermeasures are taken. StrainInfo prevents this by implementing several error detection and correction strategies. Further details of this integration strategy are given by Dawyndt *et al.* [37]. Following the terminology of Staley and Krieg, the above procedure corresponds to integrating cultures into strains. It allows representing each strain object as a separate strain passport that uses the equivalence relation of strain numbers to bundle information linked to individual cultures at a global strain level.

Meta-information about cultured microorganisms is always connected at the culture level. This is obvious for information extracted from BRC catalogs, but also holds for experimental data as discussed in the introduction of this section. As a consequence, an additional semantic integration step is needed to present the information on strain passports: descriptions about some fact or observation at the culture level need to be molded into a consensus representation at the strain level. The complexity of the aggregation procedure used to come to a consensus representation strongly depends on the fact or observation that needs to be semantically integrated. To list sequences for a given strain, strain passports simply collect all sequences linked to the cultures of the strain. Upon request, strain passports show differences in opinion about the identification of a strain by showing the union of all taxonomic names assigned by BRCs that have a culture of the strain in their holdings. However, as such conflicting information might possibly be confusing, strain passports by default show a consensus identification resulting from a majority vote selection procedure. This follows the assumption that the most probable identification is given by the taxonomic name most frequently assigned by BRCs. The absence of electronic taxonomic information makes it necessary to use the current heuristic, which in practice seems to be a workable compromise. All identifications found by StrainInfo remain available on the strain passport, by expanding a drop-down box.

BRCs typically trace the history of the cultures in their catalog by giving details on all exchanges back to the original isolate. Strain passports present a completely integrated strain history by processing this historical information that is partially documented at the culture level into a Histri using a semi-automated reconstruction algorithm [38]. Other facts and observations might require disambiguation, advanced text processing and application of domain-specific knowledge to determine a consensus representation at the strain level. Automating this is quite a challenge and a proof of concept is described in Chapter 7. Semantic integration, after all, is more than the union of descriptions at the culture level. Apart from aggregating evidence from autonomous and heterogeneous information providers that make statements about the same fact or observation, it requires all sorts of error detection and correction techniques to come to a consensus representation.

The advanced search function also makes use of the strain number equivalence relation to match strains. Strains will match a composite query if each individual search clause is matched by any of the cultures of the strain. Different search clauses can thus be matched by different cultures. This approach enriches the search for strains by combining partial characteristics described at the culture level. As a consequence, the advanced search function is able to find strains that otherwise might not have been found when searching all BRC catalogs individually.

2.2.2 Related integration initiatives

Over the years, global and local networks of microbial resource centers have taken several initiatives towards harmonization. Among those are initiatives to standardize quality control procedures, such as Common Access to Biotechnological Resources and Information (CABRI), OECD and the European Consortium of Microbial Resources Centres (EMbaRC). CABRI and EMbaRC are also define standard operating procedures. Biosafety and biosecurity regulations are governed by OECD and EMbaRC. The European Culture Collections' Organisation (ECCO) has been working on standardizing material transfer agreements. Data handling guidelines have been established the World Federation of Culture Collections (WFCC), Microbial Information Network Europe (MINE), CABRI and the OECD. As the design of StrainInfo was strongly influenced by the observed successes and shortcomings of the projects that fall into the latter category, they are discussed below.

In the 1960s the World Federation of Culture Collections pioneered the development of an international database on cultured microorganisms. The result is the World Data Center for Microorganisms[9] (WDCM) [39]. This central repository of meta-information on BRCs (and public culture collections with a non-BRC status *sensu* OECD) is maintained since 2010 at the Institute of Microbiology of the Chinese Academy of Sciences (IMCAS) and records information on 610 BRCs from 70 countries (as of June 2012). Information provided covers data on the organisation, management, services and scientific interests of the collections as well as tools for parallel research data. Registration is free of charge and each registered BRC is assigned a unique WDCM number, which is one of the criteria to be acknowledged as a member of the WFCC. Although BRCs can submit a list of taxonomic names to indicate the diversity of their holdings, information on the individual cultures is lacking. StrainInfo can thus be seen as a project building a virtual catalog on the assets of all (in theory) WDCM registered BRCs. Rather than creating its own BRC registry, StrainInfo uses WDCM numbers and detailed meta-information to identify BRCs and locate them on a map.

A first attempt to standardize the way BRCs capture meta-information about microorganisms came in the late eighties from the Microbial Information Network Europe. The outcome were two separate standards for bacteria [40] and fungi [41] describing database schemata that BRCs could use to structure their catalog information. These standards contain fixed names and definitions for database fields, along with detailed descriptions about formatting their content. Database fields were further split into a mandatory Minimum Data Set (MDS) consisting of core fields and an optional Full Data Set (FDS) for more advanced meta-information that is not always available. Definition of strict database schemata made it difficult, however, for existing databases to adopt the standard. Although legacy catalogs of BRCs still contain traces of the MINE standards even at present, they never really witnessed worldwide adoption. The lesson learned is that it has been unrealistic for BRCs to set up common database structures to harmonize meta-information on microbial cultures. A more advanced approach is to set up common machine-readable data exchange formats, so that BRCs can map their internal database structure

[9] `http://www.wdcm.org`

onto a standardized template upon data export. In collaboration with the Genomic Standards Consortium (GSC) [42], the StrainInfo team has redesigned MINE and proposed the Microbiological Common Language (MCL) [43] as a standard for electronic information exchange on cultured microorganisms. Most of the MCL field definitions are recovered from MINE, but an abstract model is employed that can be readily applied to modern representation technologies such as XML and RDF. It is currently used by StrainInfo for synchronising BRCs catalog information, data exports and web services, but is envisioned as a general framework for information exchange projects in the microbiological community.

Common Access to Biotechnological Resources and Information[10] is a project started in Europe in the late nineties [44, 45], but which also has become in disuse. Its mission statement is to set up quality standards for BRCs and impose on its members strict guidelines that could serve as a quality label. Standard operating procedures for handling different types of biological material are documented and new members were audited before they could join the consortium. A major achievement of CABRI is its development of an online search engine for querying the catalogs of all participating collections at once. CABRI is the first true catalog unification system ever put in place in the field, and featured a common interface for ordering cultures from member collections. The search engine indexes 28 catalogs covering bacteria, archaea, fungi, yeasts, plasmids, phages, DNA probes, animal and human cell lines, plant cells and viruses. Member BRCs are supposed to regularly upload exports of their catalog in a common exchange format. CABRI uses a somewhat simplified version of MINE that was analogously split into a Minimum Data Set (MDS), a Recommended Data Set (RDS) and a Full Data Set (FDS). Although promising and ambitious, the search engine never reached critical mass and was abandoned when funding ran out. Motivating BRCs to make standardized exports at regular time intervals turned out to be a critical stumbling block. In addition, a technical issue is that redundancy is introduced in search results. Searching for cultures with certain properties can yield multiple cultures of the same strain. Users have to manually inspect the `Other Collection Numbers` fields to determine which cultures belong to the same strain. This prompted StrainInfo to work with strain number equivalence information when performing searches that span across the borders of individual collections and to present search results that integrate equivalent strain numbers.

[10] http://www.cabri.org

2.2.3 Maintaining the StrainInfo index

Regular synchronisation with BRCs catalogs is necessary to cover newly deposited cultures and capture up-to-date meta-information about cultured microorganisms. Synchronisation between StrainInfo and the online catalogs of individual BRCs was initially mediated through web scraping: an indexer downloaded all catalog pages and parsed out relevant fields. This somewhat outdated technique was justified because it does not put any requirements on BRCs other than having an online catalog that can be parsed. Whereas we simply considered web scraping as the only solution that allowed for instantaneous implementation of a prototype for StrainInfo, it definitely lacks the scalability required to process over 600 BRCs/culture collections that are currently registered at the WDCM. It is extremely fragile, error prone and labor intensive [34]. If BRCs reformat online catalogs, their parser needs to be rewritten. If unforeseen parse errors are undetected, erroneous information might ruin the whole integration pipeline. What was clearly missing is a standardized data exchange format to swiftly transfer information on cultured microorganisms. But although the CABRI exchange format was adopted by the OECD as a best practice guideline for BRCs [17], it lacks the expressiveness of present-day representation technologies to be considered a mature candidate.

With more collections willing to become synchronized by StrainInfo, the web-scraping approach – initially used as a stepping stone to get StrainInfo started without the need for BRCs to make an early investment in adapting their existing IT platforms – became unsupportable in the long run. Where screen scraping was used to pull information from BRCs into StrainInfo, we switched gears towards implementing a push paradigm. We now offer the possibility for BRCs to upload (push) MCL exports of their catalogs into StrainInfo on their own pace. Existing databases that underpin the online catalogs can be easily mapped to MCL due to its adoption of legacy terms. Moreover, XML is a widely used technology with good tool support. This being said, however, a steady migration is expected from web scraping to MCL exchange as BRCs often lack in-house IT staff and find it difficult to implement automated export procedures. StrainInfo will thus support a dual push/pull strategy for indexing BRC catalogs within the foreseeable future.

Even though synchronisation based on MCL XML greatly reduces the amount of human intervention required to update and maintain the StrainInfo index following an accumulative learning approach, it remains under constant surveillance of stringent automated quality control procedures in combination with a limited amount of manual curation. Since the initial StrainInfo prototype in 2005, hundreds of thousands of strain numbers and strains have been discovered and were incrementally added to the index. While dealing with dirty (web scraping), ambiguous and noisy data (erroneous equivalence assertions made by BRCs), the error detection and correction procedures built into the indexer have proven to be essential cornerstones to keep the information provided by StrainInfo error-free [37]. It turned out that the data integration algorithm allows for discovery of inconsistencies that would otherwise be almost impossible to find. The necessity of human verification, however, always has been and will remain essential to ensure high data quality standards and to rule out cases where errors have percolated into repositories of primary data providers. As a combination of manual and automated curation, a blacklist of the errors found in BRC catalogs is maintained during operation of the equivalence integration algorithm. This prevents resolved errors to re-enter the system when continuously processing information delivered by primary data providers. Histris were designed as an additional tool to verify merges of cultures into strains, and are actively employed by users to clarify cases of strains with uncertain authenticity. It is also expected that extending the scope of semantic integration will open doors to build even more fine-grained error correction and detection strategies into the integration process.

At the time of writing, StrainInfo is indexing more than 60 BRC catalogs of which 14 are synchronized using MCL (see Table 2.1). This has merged about 670.000 strain numbers into some 290.000 strains. Long tail statistics hold: most strains were assigned only a limited number of equivalent strain numbers. In particular, for 42,3% of the strains only a single strain number was issued. Important reference strains such as type strains and strains for which whole-genome sequences are available usually have a large number of equivalent strain numbers, witnessing their broad international distribution [21]. It will come as no surprise that we observed strain passports to have page view counts proportial to the number of strain numbers assigned to the strain. Detailed and up-to-date statistics about the size and content of the StrainInfo index can be found online[11].

2.2.4 Culture URIs as globally unique identifiers

Not all strain numbers are associated with BRCs. This is the case when researchers issue their own local numbers (so-called 'researcher numbers'). Researchers introduce ambiguity by their informal usage of their own local numbers. In general, they have a simple form (e.g. `A1` or `R-4`), and by consequence, there is a high risk of

[11]`http://www.straininfo.net/stats`

WDCM number	Acronym	Full name	Country
13	ACM	University of Queensland Microbial Culture Collection	Australia
1	ATCC	American Type Culture Collection	U.S.A.
783	BCC	BIOTEC Culture Collection	Thailand
642	IHEM	BCCM/IHEM	Belgium
643	LMBP	Belgian Coordinated Collections of Microorganisms / LMBP Plasmid Collection	Belgium
59	BCRC	Bioresource Collection and Research Center	Taiwan
823	CBMAI	Brazilian Collection of Microorganisms from the Environment and Industry (Colecao Brasileira de Microrganismos de Ambiente e Industria)	Brazil
133	CBS	Centraalbureau voor Schimmelcultures, Fungal and Yeast Collection	Netherlands
807	CCAC	Culture Collection of Algae at the University of Cologne	Germany
522	CCAP	Culture Collection of Algae and Protozoa	U.K.
150	CCFC	Canadian Collection of Fungal Cultures	Canada
532	CCLM	CSIRO Collection of Living Micro-algae	Australia
65	CCM	Czech Collection of Microorganisms	Czech
883	CCMM	Moroccan Coordinated Collections of Microorganisms	Morocco
2	CCMP	Provasoli-Guillard National Center for Culture of Marine Phytoplankton	U.S.A.
885	CCT	Colecao de Culturas Tropical	Brazil
32	CCUG	Culture Collection, University of Goteborg	Sweden
500	CDBB	Coleccion Nacional de Cepas Microbianas y Cultivos Celulares	Mexico
412	CECT	Coleccion Espanola de Cultivos Tipo	Spain
639	CFBP	Collection Francaise des Bacteries Phytopathogenes	France
550	CGMCC	China General Microbiological Culture Collectio Center	China
759	CIP	Collection de L'Institut Pasteur	France
788	CIRM-Levures	Centre International de Ressources Microbiennes - Levures	France
130	CNCTC	Czech National Collection of Type Cultures	Czech
707	DMST	Department of Medical Sciences Culture Collection	Thailand
274	DSMZ	Deutsche Sammlung von Mikroorganismen und Zellkulturen GmbH	Germany
115	FGSC	Fungal Genetics Stock Center	U.S.A.
18	FRR	Food Science Australia, Ryde	Australia
779	HAMBI	HAMBI Culture Collection	Finland
195	HUT	HUT Culture Collection	Japan
190	IAM	IAM Culture Collection	Japan
589	ICMP	International Collection of Microorganisms from Plants	New Zealand
191	IFO	Institute for Fermentation, Osaka	Japan
214	IMI	CABI Bioscience Genetic Resource Collection	U.K.
567	JCM	Japan Collection of Microorganisms	Japan
806	KACC	KACC, Korean Agricultural Culture Collection	Korea (Rep. of)
597	KCTC	Korean Collection for Type Cultures	Korea (Rep. of)
296	LMG	Belgian Coordinated Collections of Microorganisms / LMG Bacteria Collection	Belgium
308	MUCL	Mycotheque de l'Universite catholique de Louvain	Belgium
816	MUM	Micoteca da Universidade do Minho	Portugal
135	NBIMCC	National Bank for Industrial Microorganisms and Cell Cultures	Bulgaria
825	NBRC	NITE Biological Resource Center	Japan
485	NCAIM	National Collection of Agricultural and Industrial Microorganisms	Hungary
797	NCCB	the Netherlands Culture Collection of Bacteria	Netherlands
3	NCIM	National Collection of Industrial Microorganisms	India
653	NCIMB	National Collections of Industrial Food and Marine Bacteria	U.K.
184	NCPF	National Collection of Pathogenic Fungi	U.K.
126	NCPPB	National Collection of Plant Pathogenic Bacteria	U.K.
154	NCTC	National Collection of Type Cultures	U.K.
134	NCWRF	National Collection of Wood Rotting Fungi	U.K.
169	NCYC	National Collection of Yeast Cultures	U.K.
591	NIES	Microbial Culture Collection	Japan
97	NRRL	Agricultural Research Service Culture Collection	U.S.A.
481	PCC	Pasteur Culture Collection of Cyanobacteria	France
124	PTCCI	Persian Type Culture Collection	Iran
829	RCC	Roscoff Culture Collection	France
192	SAG	Sammlung von Algenkulturen at Universitat Gottingen	Germany
73	UAMH	University of Alberta Microfungus Collection and Herbarium	Canada
606	UTEX	The Culture Collection of Algae at the University of Texas Austin	U.S.A.
342	VKM	All-Russian Collection of Microorganisms	Russian Federation
139	VTT	VTT Culture Collection	Finland

Table 2.1: BRCs indexed by StrainInfo.

overlaps. As these identifiers are often the first identifiers of newly isolated cultures (before deposit), they are used in publications and can also be found in catalogs. In Figure 2.1, the strain number AMMD corresponds to the original isolate and is not associated with a BRC, but has come into wide use. Ambiguity also arises when identical acronyms occur or strain numbers are reused by BRCs. For example, the LMG 2333 culture and all other LMG strain numbers used by Nilsen *et al.* [46] do not belong to the well-known BCCM^{TM}/LMG Bacteria Collection, but probably are internal numbers of the Laboratory of Microbial Gene Technology at the Agricultural University of Norway. Consequently, there are numerous cases where strain numbers are not globally unique and point to different strains in different contexts. This is highly problematic as ambiguous references introduce uncertainty about the actual material used. In addition, these identifiers are also used to refer to many other things. For example, A1 (in addition to be a paper size) is used for more than sixty different strains and R-4 (a local expressway) was assigned to nine strains. Therefore, the use of these identifiers to refer to microorganisms is highly questionable and should be avoided whenever possible. Furthermore, even the use of unique strain numbers impedes bioinformatics applications from harvesting microbiological information in an autonomous way. The additional meaning encoded in the particular form of strain numbers and the implicit context in which they are used helps researchers to determine the intended culture. Unfortunately, computers are not able to perform this disambiguation as easily as humans do, and need an explicit handle. Although the use of strain numbers may seem sufficient for current practice, it actually hinders the way towards electronic processing of microbiological information [47].

StrainInfo does not identify cultures with strain numbers, but internally uses culture identifiers ('cultureIds') instead [37,43]. CultureIds were introduced to overcome the problems with non-unique strain numbers as traditionally used in microbiology. A cultureId is basically an integer number corresponding to an arbitrary strain number. It is used to distinguish between different occurrences of identical strain numbers. For each distinct use of the same strain number, a separate cultureId is assigned. Ambiguous strain numbers can be mapped to a cultureId by making use of equivalent strain numbers or additional information such as species names. Although this needs to be executed with great care, StrainInfo for example uses contextual information to resolve a considerable portion of the ambiguous strain numbers in sequence records.

From a technical standpoint, cultureIds are not sufficient. Although it is guaranteed that cultureIds are unique identification numbers for cultures (having solved the

Strain number	cultureId	Culture URI
ATCC 53795	355608	http://www.straininfo.net/strains/355608
CIP 107266	681413	http://www.straininfo.net/strains/681413
DSM 16087	691135	http://www.straininfo.net/strains/691135
AMMD	355609	http://www.straininfo.net/strains/355609

Table 2.2: CultureId and culture URI of some example strain numbers. AMMD is a researcher number, and its culture URI makes it possible to find additional equivalent strain numbers.

ambiguity problems of strain numbers), they cannot be used as universal references to cultures as they only make sense in the context of StrainInfo. In order to make cultureIds universally usable, they need to be formatted (or packaged) in a way that computers can understand the context of the identifier. This is why StrainInfo formats them as Uniform Resource Identifiers (URIs) [48]. To do so, they are prefixed with the namespace `http://www.straininfo.net/strains/` to form a 'culture URI'. Table 2.2 contains the culture URIs of the cultures used in Figures 2.1 and 2.2. Culture URIs refer to their corresponding culture, and by extension to their implied strain, in the same way as strain numbers do. Culture URIs can be seen as an alternative representation of strain numbers, designed to better concur with the limitations of computers.

To make culture URIs useful, it is necessary to include a mechanism to retrieve associated information, i.e. making the URI resolvable and dereferenceable. This is something humans do almost unwittingly: given a strain number, an experienced microbiologist will know where to look for additional information, for example in a print catalog or online. However, this is not evident for computer systems as they rely on standardized procedures to automatically resolve (determine the location of) and dereference (retrieve) associated information. In contrast to other identifier schemes such as the Life Science Identifier (LSID) [49] and the Digital Object Identifier (DOI) [50] that have implemented their own resolution systems, culture URIs use the most widely deployed resolution system in the world – the World Wide Web (WWW). In other words, culture URIs leverage the Web to allow users and applications to find and retrieve associated information. When entered in a web browser, the culture URI forwards the user to the corresponding strain passport. Autonomous agents (applications) can request an electronic version of the strain passport using the same culture URI. Using content negotiation [51, 52], the resolver determines whether a

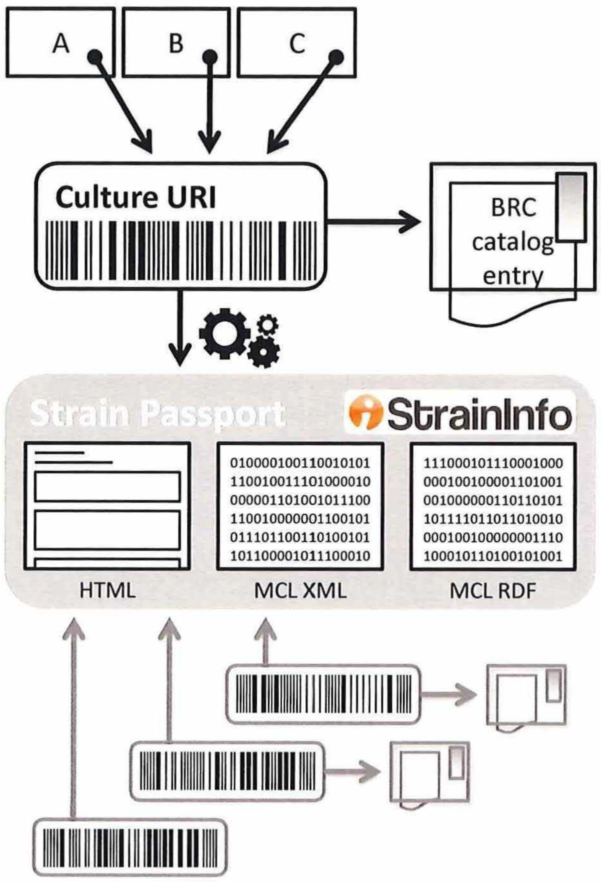


Figure 2.3: Three resources (A, B and C) refer to a strain using a culture URI. The culture URI acts as an anchor for the strain passport and strain browser on StrainInfo. The strain passport is available as a web page (HTML) and in electronic formats (MCL XML and MCL RDF), selected using content negotiation. Other culture URIs corresponding to equivalent strain numbers refer to the same strain passport. As not all strain numbers are associated with a BRC, not all culture URIs point to a catalog entry in the strain browser.

human readable (HTML) or specific electronic version (MCL XML or MCL RDF) needs to be delivered. This is illustrated in Figure 2.3. In practice, the culture URI can thus be used as a permanent URL (also known as a permalink) linking to the strain passport corresponding to the referenced culture. This makes the culture URIs tangible and understandable for microbiologists, making a strong connection with what they represent, without detracting from their technical intentions. Culture URIs are not intended to replace strain numbers, and users should be able to stay with the identifiers they have experience with. However, computer applications should always use URIs internally. If an application is able to handle culture URIs, it is able to address all known cultures and retrieve information in an electronic format. Showing the strain number as the label of hyperlinks, as it is done on the Web, combines the best of both worlds.

The electronic information is available in two representation formats using MCL terms, i.e. XML and RDF. XML is a well-understood file format with many available tools and MCL XML files are already used for synchronisation with BRCs (catalog schema, [43]). In addition, the microbial information is also available as MCL represented using RDF files [53, 54]. RDF has a fundamentally different abstract model which makes it better suited for distributed data integration. RDF uses URIs to refer to entities, which allows to refer to other data sets formatted using RDF, effectively building a Web of Data. These practices comply with the conventions of the Linking Open Data project[12] and therefore make StrainInfo available as Linked Data [55]. This solution thus addresses both interoperability and identity problems [56], while making use of standard technology. Being linked to a wide wealth of data opens the possibilities for analysing new relations on a large scale.

2.2.5 Problems of alternatives to culture URIs

It might be tempting to think that it would be more convenient to introduce one *strain* URI per strain passport instead of maintaining multiple equivalent culture URIs. However, this is not the case, and in fact problematic. Strain passports are dynamic as they might change whenever StrainInfo acquires additional information. If, as illustrated in Figure 2.4, a lineage of cultures is identified as belonging to a separate strain (e.g. in the case of a contamination), those cultures need to be moved to a new strain. This would require that all linking resources need to replace their strain URI link by one of the two new strain URIs. However, using the original strain URI, it is impossible to determine which lineage of the strain was

[12]http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

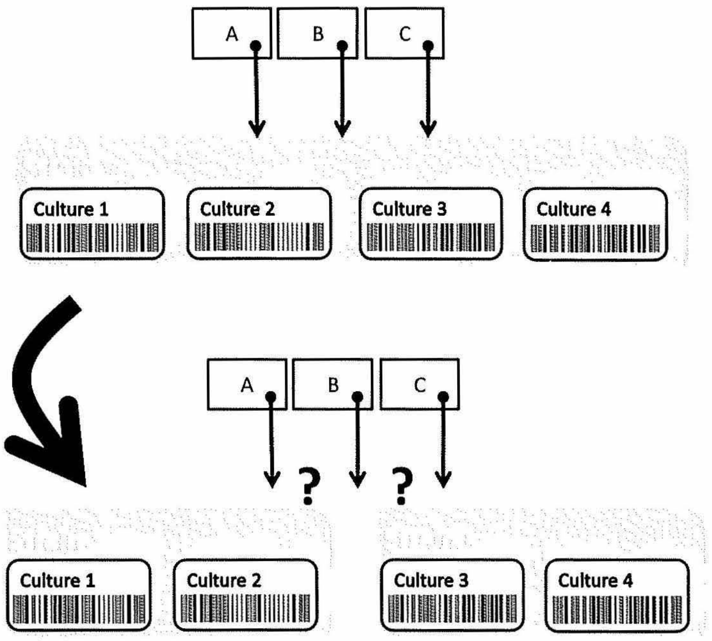


Figure 2.4: Situation where resources directly link to strain passports. If new information becomes available that proves that cultures belong to different strains and two new strain passports are created, resources need to change their links to the new strain passports. Based on their original link, it is impossible to determine which new strain passport to link to.

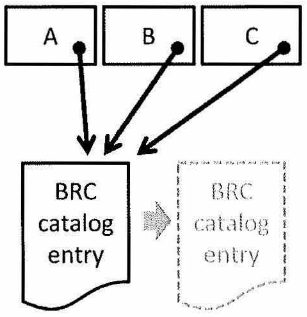


Figure 2.5: Situation where resources link directly to BRC catalogs. If a catalog entry changes location, all resources pointing to that entry need to update their links.

intended. Culture URIs do not have this issue as they automatically point to the strain passport of the strain they belong to. Generally speaking, strain passports continue to expand as the result of discovering additional equivalent strain numbers, missing links between two previously separated strains, or new semantic information in resources. If a strain passport expands or changes, it should get a new strain URI because new information might change aspects of that strain which could result in significantly different downstream results (e.g. a different semantic integration result could trigger a different classification in a downstream application).

One might also argue that the current URLs of BRC catalogs could serve as culture URIs. However, this is not feasible as many strain numbers are not associated with BRCs. This is the case for researcher numbers, historical strain numbers whose corresponding collection has disappeared or changed its name and other strain numbers that do not have an online catalog entry. Technically, some BRCs do not have a separate page per catalog entry (but have for example a page per species) and might need additional addressing in the page. It would also imply that BRCs could never change the link structure of their online catalog, strongly limiting the possibilities of online catalog and website upgrades. Nevertheless, in practice, the link structures of BRC catalogs do change, possibly rendering existing links on the Web unusable. Therefore, StrainInfo also makes it possible to use the culture URI to create a permalink that resolves to the corresponding BRC catalog page in the strain browser by appending `/browser` to the culture URI. Links made from other resources or personal bookmarks to BRC catalogs using these permalinks no longer depend on the BRC catalog website structure. When a BRC updates its website and changes its internal page structure, StrainInfo will internally change its catalog entry resolver, keeping its permanent URLs pointing to what is expected, and, by consequence, keeping the link or bookmark valid. This is an elegant approach as from ga global perspective only one link needs to be updated per catalog entry, in contrast to the situation in Figure 2.5, where all linking resources need to update their link. Moreover, as StrainInfo's catalog entry resolver is rule based (see Section 3.2.1), the change is performed efficiently by updating the BRC catalog URL template. The catalog entry resolver is kept up-to-date by automatic verification of the BRC catalog page links at regular time intervals.

2.2.6 Ghost cultures

It is possible to find strain numbers that yield BRC catalog error pages when viewed in the strain browser. This phenomenon is called 'ghost cultures': strain numbers that are discovered by StrainInfo, but that are, although seemingly valid, not listed in the corresponding BRC catalogs. This is different from strain numbers that due to their nature have no associated BRC catalog entries (e.g. researcher numbers). Although the strain numbers have been valid at some point in time, the corresponding material is no longer available from that particular BRC. In that case, most BRCs remove the strain numbers and corresponding information from their catalog. This often happens when a culture cannot be revived or when multiple cultures of the same strain have been deposited in a particular BRC (and duplicates have been discarded). If a culture cannot be revived, the strain is mostly accessioned again under an alternative number. It can also be the case that the culture is not publicly available from the BRC, but that it will become available in the future, for example, if the BRC has a quality control backlog or if the culture has been deposited explicitly as non-public (e.g. safe and patent deposits).

Although it is commonplace that BRCs omit catalog entries of unavailable strains, it is actually problematic. It is important to note that a catalog entry itself is always valid, even if the culture is not available from the BRC. The catalog entry contains the knowledge of that culture at the time it was last updated. Analogously, publications are snapshots of scientific insight at a certain point in time. Even if the publication becomes obsolete, access to the original publication and biological material is essential to fully understand that snapshot. If a strain has been referenced in a publication and if it is removed from the original catalog, the reference becomes useless. However, in this case, StrainInfo can be used to retrieve equivalent strain numbers, owing to the accumulative learning nature of the StrainInfo index. Ghost cultures are never removed from the index after BRC catalog entry updates.

2.3 Conclusion

Although the bioinformatics community is building large-scale infrastructure focusing on experimental results, it is also important to keep track of and document the microbiological material from which the results were obtained. StrainInfo is the first initiative that effectively performs both equivalence and semantic integration of microbial information, and bundles this information in strain passports. This allows users to start from the biological material itself and discover downstream information and experimental results.

The culture URI was introduced as a universal anchor for microbial information and references the strain passport associated with the culture. By dereferencing the culture URI, the information from the strain passport can be obtained in both human-readable and electronic form. The right format is selected using content negotiation. For users, this makes the culture URIs usable as permalinks to an attractively designed strain passport. In addition, the strain browser allows users to view the underlying BRC catalog entries. Autonomous agents consume an XML or RDF representation of the passport based on the Microbiological Common Language (MCL).

The StrainInfo platform is envisioned to proliferate microbiological information, aggregated from a wide set of distributed resources. StrainInfo is open and publicly available, accessible online through a web application and web services located at `http://www.straininfo.net`.

This chapter is based on:

B. Verslyppe, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. StrainInfo introduces electronic passports for microorganisms. *PLoS ONE*, 2012. (submitted)

3 The StrainInfo platform software architecture

StrainInfo as it is available today is the result of a significant growth process. The StrainInfo index evolved from a handful of BRCs to more than sixty BRCs. As the scale of StrainInfo increased, additional functionality was added to the original web application. Initially launched publicly in February 2007, the legacy version of the web application unfortunately reached the point where new additions would require a significant amount of development effort only one year later. As the project had grown organically, the project did not employ rigid software engineering practices from the start. There was no consistent separation of concerns: web and representation code was mixed with core integration algorithms, probably the result of shortcuts and attempts to improve performance during initial prototyping. The web application was built as a collection of individual web pages, without structure offered by a web framework. As a result, similar functionality was repeated at different places (both in the web application and the database).

With the passport/browser concept and the feature wish list in mind, the development of a new codebase was started in September 2008. Features such as web services and personalisation would require a significant amount of effort to restructure the existing code, and therefore it was decided to build a completely new architecture instead of refactoring old code. However, it was not feasible to start completely from scratch. Not only is the StrainInfo index (i.e. the data itself) the result of numerous incremental executions of the integration algorithm, it is also the result of a significant amount of manual verification and curation. Starting from scratch would mean that this work would need to be redone. Additionally, many of the core algorithms (such as the equivalence integration algorithm) are hand crafted to support many possible variations and exceptions in input data. Therefore, it was decided to retain these labour intensive parts, and focus on re-engineering the parts necessary for efficiently building a modern, extendable and maintainable web application. The new codebase uses legacy stored procedures where relevant and also contains rewritten (but functionally identical) methods to allow certain operations to be done without calling the database. The web application itself was completely written from scratch. About one year later, the results of this development effort was launched publicly. The new web application was released on August 24^{th}, 2009 and test run in parallel with the legacy StrainInfo until the latter was taken out of service on October 14^{th}, 2009. As of today, the StrainInfo logo on the web application still bears the tag 'beta' to indicate continuous development and improvement of StrainInfo.

The new codebase is referred to as the 'si2 codebase' and is written using the Java 1.6 platform. This allows to write code for a multitude of applications, ranging from traditional web applications to web services and stand-alone tools. The codebase was conceived as a common starting point for new experiments with StrainInfo. It provides reusable access to StrainInfo data and functionality, lowering the overhead for experimental code. Additionally, having access to a full-fledged programming language with an extensive collection of freely available libraries allows for complex experiments to be set up with the convenience of sticking to a mature platform.

3.1 Three tier architecture

StrainInfo is implemented using a multitier architecture. In multitier architectures, different concerns in an application are architecturally divided into separate layers or tiers. Conceptually, the architecture contains three tiers, i.e. a data tier, a middle tier and a presentation tier (see Figure 3.1). Each layer adds functionality

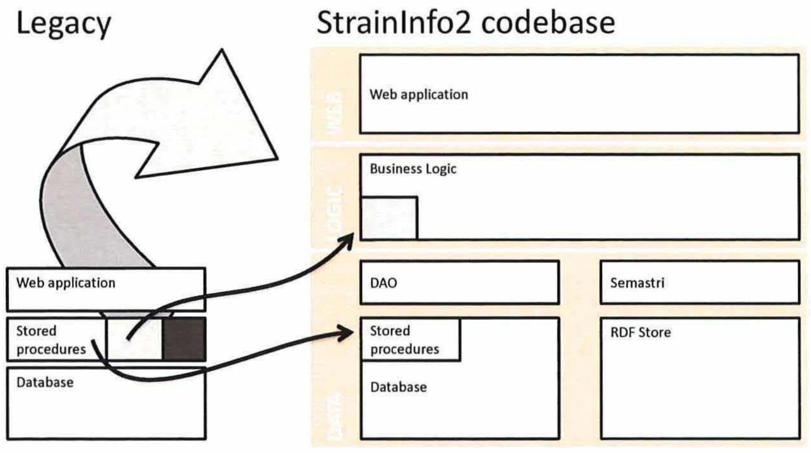


Figure 3.1: The multitier architecture of the si2 codebase and its relationship with the legacy StrainInfo architecture.

by building on the previous layer. This is done by exposing the functionality through defined interfaces. Layers communicate solely using these interfaces, which allows to replace a layer with a new implementation without impacting the other layers. For example, this allows to migrate to a different database by rewriting only the database-associated layer, without affecting the other layers. This is particularly useful when moving towards a totally different database concept such as sharding or a file based repository. The functionality of the three tiers is divided as follows:

Data tier Layer containing the database itself and the code responsible for information exchange with the database. This layer communicates with the database tables and calls legacy stored procedures.

Middle tier Also known as the logic tier or business logic. Uses the data from the data tier, and contains all functionality to handle and modify this data. Functionality includes parsing and handling of strain numbers, URL generation, the implementation of the Microbiological Common Language (MCL) and XML generation functionality.

Presentation tier Layer containing the web application and stand-alone tools. In the context of a web application also called the web layer. Adds a user interface to the functionality offered by the middle tier.

Figure 3.1 shows how the legacy StrainInfo is related to the three tier architecture. In the legacy architecture, the database and what could be regarded as business logic are tightly coupled to the DBMS. Next to the database tables themselves, stored procedures (written in PL/SQL) are stored and run in the Oracle database. Unfortunately, this code is minimally structured and hardly documented. In addition, PL/SQL does not seem to be a practical choice for some features such as XML generation, web service back-ends, support for MCL or any computationally complex operation. In addition, some of that functionality (the dark grey block in Figure 3.1) actually is presentation tier code that was incorporated into the stored procedures, probably in an attempt to optimise the web application. These stored procedures format the output of other stored procedures (or sometimes depending on a flag, even their own output) as HTML code. This code is ignored by the si2 codebase. Some functionality (the light gray block in Figure 3.1) is important and was rewritten as Java code to have that functionality without needing to do a roundtrip to the database for each call. This includes the `CatalogLinkOut` system which is responsible for generating URLs of BRC catalog entries, and the `StrainNumber` class which formats, parses and models strain numbers (see Section 3.2.1). Some legacy stored procedures (mainly all procedures for equivalence integration and processing of the Histri XML format) are not rewritten and remain part of the database. Although, strictly spoken, those stored procedures perform business logic, we consider them part of the data tier. This is explained by the database access that is necessary to call them. If these procedures would be rewritten in the si2 codebase, the majority of the new code would be placed in the middle tier, as the actual logic (i.e. the data manipulations) belongs to the middle tier. Some of their code would move to the data tier, i.e. all code that accesses the database tables to collect all necessary data (to the extent that is not already existing). The presentation tier has access to these methods through its interface on the middle tier.

The legacy web application itself was written in JavaServer Pages (JSP), without using a framework, tag library or consistent approach. As a result, this *spaghetti* *code* is almost impossible to understand, maintain and extend. Therefore, nothing was recycled from the web application. This also motivated rethinking all usability aspects, which eventually led to the passport and browser concepts.

3.2 StrainInfo platform Maven modules

The si2 codebase was designed from the ground up to be easy to understand, modular and portable. Modularity and portability are pursued by using Apache Maven[1] [57]. Maven is a build tool that is able to perform dependency management. The POM (Project Object Model) file, central to each Maven module, can be compared to a Makefile. It declaratively defines all different aspects of the module's build, which – in Maven terminology – results in an artifact with a qualified name and a version. External libraries defined in the POM file are automatically resolved, downloaded and added to the class path at compile and run time. This makes the build machine and platform independent as the code can be checked out on a new machine or development environment and can be built and run without having to worry about including required libraries to the class path. Maven embraces the idea of *convention over configuration*, i.e. Maven provides default values for the project's configuration and assumes a standardized project layout.

The si2 codebase consists of several Maven modules, more or less split according to function. This is shown in Figure 3.2. All modules are bundled in a parent project (`platform`) that contains common attributes for all modules in the project. In addition, this also allows building all modules with one invocation of Maven. All modules make use of the functionality offered by the `core` module, which contains the data and middle tier. Both the `webapp` module (containing the web application), `tools` module (containing stand-alone tools) and `sitemap` module (generating the sitemap) belong to the presentation tier. The `experiments` module is an overlay of the `webapp` module, and extends the web application with some experimental features. In what follows, a modular breakdown of the platform is described.

3.2.1 Core module: data and middle tier

The data and middle tier have been joined into the core module. This is mostly because of practical reasons, as the data tier only consists of mapping code in a limited number of packages. The Inversion of Control container (Dependency Injection) of

[1]`http://maven.apache.org/`

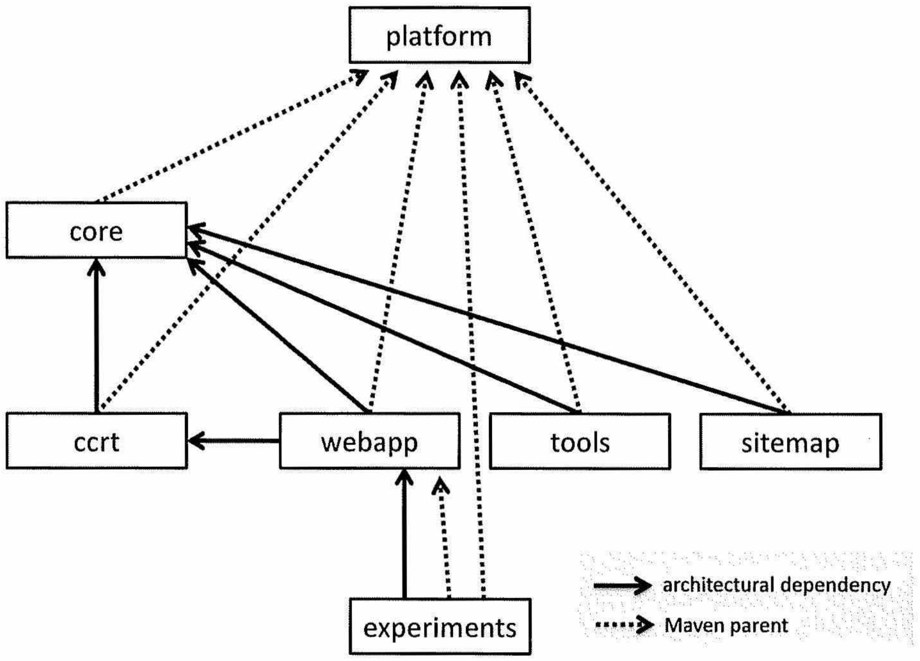


Figure 3.2: Maven modules and their mutual dependencies in the si2 codebase.

the Spring Framework[2] is used to introduce loose coupling of implementation objects [58]. As a rule, code is implemented against interfaces, and the dependency injection (DI) mechanism is used to inject the actual implementation.

Data tier

Access to the database is performed using the Java Persistence API (JPA), using the Hibernate implementation. Database Access Objects (DAOs) contain methods that retrieve data from the database. Put simplified, each method can be seen as a query that returns data objects or performs an update to the database. All DAOs are defined in the package `net.straininfo2.jpa` which contains interfaces, and the actual implementations are held in the `jpa` subpackage. Switching to another underlying database paradigm can be accomplished by creating a new set

[2]`http://www.springsource.org/`

of implementations for the DAO interfaces.

The actual object-relational mapping (ORM) entities are held in the package `net.straininfo2.entities`. Mappings held in the `legacy` subpackage are mappings to tables that are used in the legacy StrainInfo. As a consequence, these mappings are tightly bound to the database schema, which results in a mapping that is more complex and less elegant compared to standard ORM practices. The underlying tables consist of the core StrainInfo tables, on which the legacy stored procedures also operate. Adapting the schema of those tables would require refactoring the legacy stored procedures, something that was determined in advance as too cumbersome. All other entities are new entities that were created to serve new features of the si2 codebase. Therefore, the corresponding tables reside in separate database namespaces (`si2`, `si2_sync` and `si2_mysi`). Some important legacy entities include:

Culture Entity representing a culture identified by a strain number. The `Culture`[3] class is a mapping of the `si2.cultures` view in the database. It is the central entity in StrainInfo as most information is linked at the culture level. The `si2.cultures` view is a slightly filtered version of the table `catalog.cultures`, on which the original legacy stored procedures operate, but contains non-public cultures. `Culture` entities essentially model a strain number, and therefore the concepts strain number, culture and the class `Culture` are strongly related and often can be used interchangeably in the context of the si2 codebase.

An important aspect is the uniqueness of strain numbers. A `Culture` (or a strain number) is considered unique if:

- The acronym (`getAcronym()`) is not `null`, i.e. the strain number has an acronym.
- The acronym itself is considered to be a unique acronym (`Acronym.isUnique()` is `true`).
- The strain number number component (`getStrainNumberNumber()`) is not `null`, i.e. the strain number contains digits.

Unique strain numbers are considered valid globally unique identifiers. In general, they correspond to BRCs where they belong to a structured numbering

[3]`net.straininfo2.entities.legacy.Culture`

scheme, which conceptually guarantees world-wide uniqueness[4]. The notion of a unique strain number is used in the legacy stored procedures (equivalence integration) and also to determine whether `Cultures` can be unambiguously identified (i.e. resolving vs. searching). In addition, unique `Cultures` often have formatting information attached to their `Acronym`.

The status field (`getStatus()`) contains information about the validity of the `Culture`. A strain number can be invalid when it for example contains inconsistencies as the result of unwanted crawling artifacts or typos. In that case, it is made invalid (as a rule, `Cultures` are never removed from the database). When the `Integer` status value is greater than zero, the `Culture` is considered invalid. Currently, only the values `0` and `1` are used, but in the future other status values can be introduced, to discriminate between different reasons why cultures are invalid.

All additional information is attached using the `CultureTriple`[5] entity. An exception is the species name as given by the BRC, as this information is used by the legacy stored procedures. The legacy code also defined additional columns on the `si2.cultures` view, but these are not mapped into the entity and ignored by the si2 codebase.

The equivalence integration integrates `Cultures` into `Strains`. The actual strain equivalence relation is stored in the `si2.cultures` view by giving equivalent `Cultures` the same `strainId`.

Strain Entity representing the strain concept, binding equivalent `Cultures` together. The corresponding table (`catalog.strains`) is mapped using the `Strain`[6] class. It does not contain any information with regards to equivalence of `Cultures`, but is a holder for the `strainIds` and contains some meta-information used by Histri. Other than that, `strainIds` do not have a real value and should not be used as an external or stable identifier, as they have no permanent binding with `Cultures`. In practice, the ORM mapping proves to be convenient, as for example the `getCultures()` method gives access to all `Cultures` of the `Strain`.

[4]However, from a computer science perspective, there is an important difference between true, technically enforced or guaranteed, global uniqueness and something that is *considered* to be globally unique. If the latter proves to be not truly unique, it can yield unexpected and unwanted behaviour in algorithms.

[5]`net.straininfo2.entities.CultureTriple`

[6]`net.straininfo2.entities.legacy.Strain`

HistriLog A `HistriLog`[7] object is persisted for each Histri save action. The entity contains some meta-information, but most content is saved in an XML CLOB (Character Large Object, i.e character data stored in a separate location referenced in the database table). Upon saving in the Histri editor, the XML CLOB is processed by a legacy stored procedure, and is stored for later reference. As access to this XML CLOB is only necessary in a few limited situations and negatively affects performance in cases where access is not necessary, a trick is used to lazily load the CLOB. Hibernate does not support lazy fetching of properties, but does support lazy fetching of join relations. Therefore, a second entity (`HistriLogXml`[8]) that only maps the CLOB on the same table and that is mapped as join relation is used to circumvent this limitation.

Middle tier

The middle tier contains business logic. This includes:

- The package `net.straininfo2.entities.views` contains Data Transfer Objects (DTOs). These are unmanaged (i.e. not mapped to the database and therefore not managed by the ORM container) objects which are immutable. They are used as the communication objects between the middle tier and the presentation tier, and therefore are returned by middle tier services. Using managed entities in the presentation tier poses risks of unintended changes to the database, for example particular situations where the web application allows users to insert data into fields retrieved from the database. Many of these DTO objects also have additional methods that help presentation code to visualise or browse complex entities (e.g. indices enabling iteration over orthogonal aspects of the data structure).
- The `StrainNumber`[9] class models, parses and formats strain numbers. A strain number is composed of an acronym, a number, a postfix and a batch number[10]. The acronym of a strain number consists of all characters before the first digit of the strain number. All subsequent digits constitute the number component. The postfix component spans from the first non-digit

[7]`net.straininfo2.entities.legacy.HistriLog`

[8]`net.straininfo2.entities.legacy.HistriLogXml`

[9]`net.straininfo2.logic.StrainNumber`

[10]The batch number is only used with LMG strain numbers and is a relic of the early days of StrainInfo, where it was an extended version of the LMG catalog.

character after the number component until the end of the strain number. It is possible that the postfix contains digits, for example for strain numbers that contain a structure consisting of two numbers separated by a dash or dot. In that case, only the first number is regarded as the number component. In addition, it is possible that the acronym, number or postfix component is empty. Strain numbers are stored in their parsed form on the `Culture` entity.

Using the `parse(String)` method, this class parses a strain number to its components. It is the Java equivalent of the legacy stored procedure `catalog.culture_pkg.parse_strain_number` and returns exactly the same result in virtually all situations. There are small differences with the original code (mainly bug fixes of the original code and the removal of legacy support for internal LMG strain numbers) that are documented in the Java file. An interesting difference with the legacy stored procedure is that this implementation saves the number component as a `String`, as opposed to the legacy stored procedure and database table, handling the number component using the `NUMBER` data type which results in leading zeros to be automatically removed. However, as some number components correspond to years (e.g. 09), omitting leading zeros is not a desirable behaviour. Therefore, in order to be able to easily adapt the si2 codebase to natively support number components with leading zeros, the number component is handled using the `String` data type. The Java method currently explicitly removes leading zeros to emulate the exact behaviour of the legacy stored procedure.

Strain numbers belonging to BRCs often have standardized formatting (e.g. capitalised acronym, a space or a dash between the acronym and the number part,...). However, in many situations, strain numbers are encountered or stored without correct formatting. Therefore, this class is able to apply the standard formatting to strain numbers based on formatting information using the `format(Culture)` method. As this depends on the BRC, this formatting information is stored in the `Acronym` of a `Culture`. This corresponds to the legacy stored procedure `catalog.culture_pkg.compose_strain_number`. Formatting is required for all external use of strain numbers (e.g. in the web application).

- The `CatalogLinkOut`[11] system is responsible for generating the BRC catalog entry URLs, an essential part of the strain browser. `CatalogLinkOut` is a rule

[11]`net.straininfo2.logic.url.CatalogLinkOut`

engine that generates the BRC catalog entry URL based on a strain number. All rules are defined in a configuration file[12]: each UrlGenerator element corresponds to a URL that is generated for a set of strain numbers. URL generation rules (implementations of the UrlGenerator[13] interface) are diverse, as depending on the BRC catalog URL structure, different components are needed to form the URL. Some BRCs use the strain number or a reduced form hereof, while others use internal identifiers (e.g. record numbers) which are stored by StrainInfo or need a decision based on the strain number structure. Each UrlGenerator rule contains a pattern to which the strain number must match (implemented using StrainNumberPattern[14]). Generating a URL for a strain number therefore corresponds to iterating over all rules, until a strain number matches a StrainNumberPattern, after which the associated URLGenerator.generateURL(Culture) method is executed.

Most rules perform substitution of parameters in a predefined URL template. This default behaviour is implemented in TemplateURLGenerator[15]. The class optimises search and replace for existing patterns in the URL template by determining on construction which patterns occur in the configured URL template. In addition, this class also supports returning null (meaning that no URL could be generated) when internal identifiers are null and adding zero padding of the strain number number component. However, more complex rules also exist. For example, CBSURLGenerator[16] generates the URLs for CBS strain numbers. CBS (Centraalbureau voor Schimmelcultures) strain numbers originate from multiple independent catalogs, which can be determined based on the number range. In addition, CCMURLGenerator, CIPpURLGenerator and VTTURLGenerator generate URLs for CCM, CIP p and VTT strain numbers respectively.

- LinkOut[17] generates links to external resources other than BRC catalogs and must not be confused with CatalogLinkOut. The latter requires strain number matching to select the right URL generation rule, while LinkOut only generates URLs. A URL base is stored in the enum, and is combined with

[12]catalogurlgenerator.xml
[13]net.straininfo2.logic.url.UrlGenerator
[14]net.straininfo2.logic.url.StrainNumberPattern
[15]net.straininfo2.logic.url.TemplateURLGenerator
[16]net.straininfo2.logic.url.CBSURLGenerator
[17]net.straininfo2.logic.LinkOut

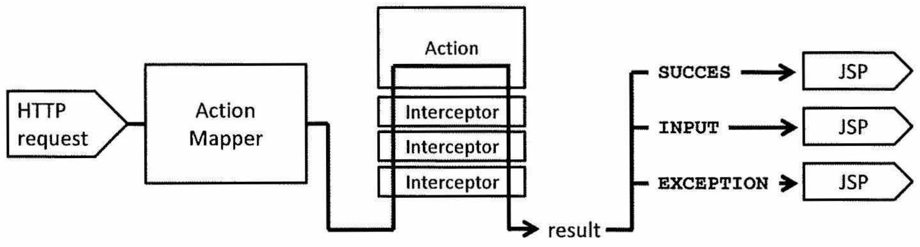


Figure 3.3: Overview of the workflow of a HTTP request handled by the Struts2 framework.

the external identifier to form the URL. This enum is also used to generate external URLs for StrainInfo passports and browsers.

3.2.2 Webapp module: web application (presentation tier)

The presentation tier includes the StrainInfo web application and some stand-alone tools that give access to data and enables maintenance of the data. The web application is contained in the Maven `webapp` module.

Overview

The web application is built using the Apache Struts2[18] framework, which is a so-called Model-View-Controller (MVC) web framework [59, 60]. The framework, of which a simplified model is shown in Figure 3.3, handles the process of receiving the request, performing the necessary tasks or database access and generating the response in a structured way. Central to Struts is the concept of `Actions`. Each HTTP request corresponds to an `Action` that performs all database access and business logic necessary to render the response to the user. The response itself (which is often, but not necessarily, HTML), is generated by a dedicated engine, independent of the `Action` that is strictly focused on logic. The response therefore must be seen as a view of the `Action`. Struts supports different templating engines, including JSP, which is used by the StrainInfo web application. If using JSP, the Struts tag library can be used to obtain fields from the executed `Action`, for example to render data that was fetched from the database.

[18]`http://struts.apache.org/2.x/`

When Struts2 receives an HTTP request, the framework uses the URL of the request to determine the `Action` that corresponds to that URL. This is performed by `ActionMapper`[19], based on a configuration file. Instead of directly executing the mapped `Action`, Struts2 first applies `Interceptors`[20]. Interceptors perform common tasks that need to be done before (or after) the execution of an `Action`. Examples include opening a database session, parsing the query portion of the URL, validating input parameters, looking up user credentials of a signed-in user, or after the `Action` is completed, adding a record to the access log or ending the database session. Often multiple interceptors are combined; this can be visualised as a stack or as concentric shells surrounding the `Action`. It is possible to configure the interceptor stack for each individual `Action`, but in practice `Actions` with the same interceptor stack are grouped in the configuration file.

If the `Action` finishes, Struts2 starts the view phase of the request. It is possible to configure multiple results for one `Action`. Each result corresponds to a way of rendering the view of the `Action`. Often this is a HTML page, but it can be very diverse: an XML file, a rendered PDF or PNG, or a HTTP redirect[21]. Based on the result returned by the `Action`, the appropriate result handler is selected and executed. By default, `Actions` return the `SUCCESS` result, which results in the default result view to be rendered. However, in certain situations an `Action` might return a different result: for example, an `Action` that performs a search function might have a default result which renders the list of search results, and a special redirect result which redirects the user if there is only one search result. Although developers are free to use their own defined results, Struts2 contains a couple of predefined results. The mechanism of using multiple results is often used in conjunction with interceptors. Interceptors can decide to not further execute the remainder of the interceptor stack and the action, and immediately return a result themselves: a practice called 'short-circuiting' the `Action`. A typical example of this is input validation. Based on the configuration of an `Action`, the input validation interceptor validates the value of the properties of the query string. If the validation fails (for example, a property is missing or does not have the correct format), it will immediately return the `INPUT` result. This result is used by convention to indicate that the input parameters are wrong and the `Action` can not execute normally. This is often the case when users enter invalid data in a HTML form. In that case, the user is returned to the form page, and warnings are added along the form fields

[19] `org.apache.struts2.dispatcher.mapper.ActionMapper`
[20] `com.opensymphony.xwork2.Interceptor`
[21] Such as `HTTP 302 Found` or `HTTP 301 Moved Permanently`.

that contain invalid data. Results are configured on the `Action` level, but it is also possible to configure default results for all or groups of `Actions`. This is typically done for the `EXCEPTION` result. The exception interceptor catches exceptions thrown by `Actions`, and returns the `EXCEPTION` result which leads to a suitable 'something went wrong' page.

Strain browser request structure

The strain browser poses a number of technical difficulties that might be unexpected at first sight. First, the strain browser shows an overlay panel on top of the underlying BRC catalog entry page. Second, not all strain numbers have an associated BRC catalog entry. Third, some BRC catalog entries are only reachable after submitting a form on the BRC catalog web site. Fourth, for a limited number of BRCs catalogs, it is necessary to open a session or to select a database before the actual post can be performed.

As a maintainable and extensible solution, the strain browser was implemented as shown in Figure 3.4. The user enters the strain browser via the standardized RESTful URL, which returns the page containing the overlay panel of the strain browser. In order to be able to offer an overlay panel, the actual BRC catalog page is loaded in a full-size iframe. In addition, this has the advantage that the RESTful URL of the strain browser is shown to the user.

The web client automatically loads the iframe content by requesting the iframe source URL. This URL also belongs to the strain browser implementation, and is a non-standard execution method of the `StrainBrowserAction`. This method determines, based on a cultureId, whether there is a BRC catalog entry associated with the strain number. If not, it returns the `nocollection` result, which results in Struts2 rendering a warning page listing all equivalent strain numbers that have a BRC catalog entry. If the strain number has an associated BRC catalog entry, the method returns a `HTTP 302 Found` redirect to the actual BRC catalog entry. The BRC catalog entry URL is generated by the `CatalogLinkOut` system.

Some BRCs have a BRC catalog entry that cannot be directly addressed with a URL. Often, it is necessary to perform a search on the BRC catalog, which gives access to the catalog entry, and the entry is returned as the result of submitting a form. Some BRC catalogs only accept forms to be submitted using HTTP POST, as opposed to HTTP GET which would allow to use a regular URL template. In order to be able to include these BRC catalogs, the strain browser contains a mechanism to transparently give access to these BRC catalog entries,

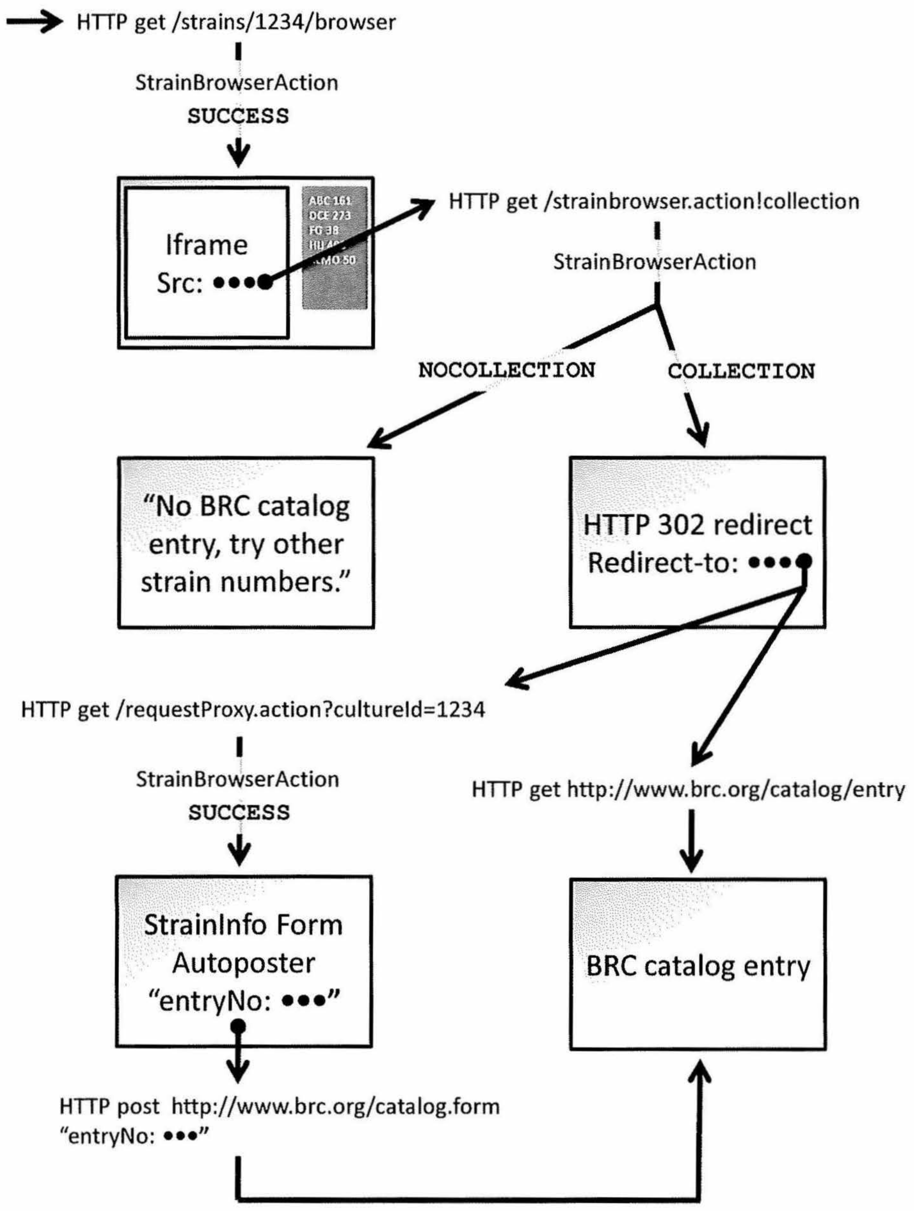

HTTP get /strains/1234/browser
StrainBrowserAction
SUCCESS
Iframe
Src: ●●●
ABC 161
DCE 273
FG 38
HTTP get /strainbrowser.action!collection
StrainBrowserAction
NOCOLLECTION
COLLECTION
"No BRC catalog entry, try other strain numbers."
HTTP 302 redirect
Redirect-to: ●●●
HTTP get /requestProxy.action?cultureId=1234
StrainBrowserAction
SUCCESS
HTTP get http://www.brc.org/catalog/entry
StrainInfo Form Autoposter
"entryNo: ●●●"
BRC catalog entry
HTTP post http://www.brc.org/catalog.form
"entryNo: ●●●"

Figure 3.4: Overview of the strain browser based on the complete request structure. URLs starting with slash are requests to the StrainInfo web application.

without requiring users to submit forms themselves. This is done using the trick shown in Listing 3.1: StrainInfo generates a form that is compatible with what the BRC catalog expects (using the necessary parameter names and values) and uses JavaScript to automatically post the form. This is almost invisible to the user, who only sees a white page for a short period of time. The class that generates this response is PostParametersRequestProxy[22]. All HTML templates are stored and formatted using ResourceBundle system in the configuration file si2-requestproxy.properties.

Listing 3.1: JavaScript-based automatic form submitter.

```
<html>
<head>
  <title>StrainInfo2 form POSTer</title>
<head>
<body onload="document.catalog.submit()">
  <form name="catalog"
        action="http://www.brc.org/catalog/search.form"
        method="post">
    <input type="hidden" name="num" value="1234">
  </form>
</body>
</html>
```

The RequestProxyAction[23] is able to hand over the generation of the HTTP response to an implementation of the RequestProxy[24] interface. PostParameters-RequestProxy is one of those implementations that belong to the RequestProxy subsystem. Although the number of BRC catalogs that do not offer RESTful access to their catalog entries has been decreasing, this is still used by four BRCs: CBS ('AC' strains only), NCCB, NCIMB and UAMH. Two other BRC catalogs (NRRL and NCAIM) require even more complex access, which is implemented in their own RequestProxy.

The catalog of the Agricultural Research Service (ARS) Culture Collection[25] (also known as the NRRL Collection) needs the opening of a session where the

[22]net.straininfo2.logic.url.PostParametersRequestProxy
[23]net.straininfo2.actions.RequestProxyAction
[24]net.straininfo2.logic.url.RequestProxy
[25]http://nrrl.ncaur.usda.gov/

catalog database (yeast or prokaryote) is selected before the actual BRC catalog entry can be loaded. This is implemented in NRRLRequestProxy[26] which returns a page that contains an iframe that loads the session opening URL. If that session page is loaded, the iframe page is forwarded to the actual BRC catalog entry using a JavaScript forward that is triggered by the onload argument.

The catalog of National Collection of Agricultural and Industrial Microorganisms[27] (NCAIM) uses a similar mechanism, but with the added difficulty that the catalog makes use of frames. All functionality is provided by NCAIMRequestProxy[28]. Loading the frameset of the catalog is necessary to open the session. However, this only shows the navigation frame, and the content frame is left blank. After loading the frameset page in an iframe, the client is forwarded to a new NCAIMRequestProxy page which contains a slightly modified version of the frameset used by NCAIM, with the actual catalog entry URL set as the frame content source. The additional parameter catalog is used to distinguish between the two different pages returned by this RequestProxy. The catalog entry URL is based on internal identifiers stored in the StrainInfo index.

ForwardRequestProxy[29] is associated with the Histri domain knowledge (HDK) implementation. HistriDomainKnowledge[30] models the Histri domain knowledge (see Figure 4.2) in the si2 codebase. The domain knowledge encompasses changes to strain numbers (such as acronym changes) as the result of institutional changes (see Section 4.1.2). As a result, some obsolete strain numbers can be translated into current strain numbers that have a BRC catalog entry. ForwardRequestProxy forwards the user to the BRC catalog entries corresponding to the current strain numbers. In addition, it shows a yellow warning bar on top of the page to indicate the forward.

The use of RequestProxy is also configured in the CatalogLinkOut configuration file. CatalogLinkOut automatically generates the internal RequestProxy URL using the RequestProxyURLGenerator[31]. In addition, all ForwardRequestProxys corresponding to HistriDomainKnowledge institutional transfers are automaticaly added. The complexity of CatalogLinkOut is necessary to support a limited number of BRCs with old-fashioned URL structures. Since the original launch of

[26] net.straininfo2.logic.url.NRRLRequestProxy
[27] http://web.uni-corvinus.hu:8089/NCAIM/index.jsp
[28] net.straininfo2.logic.url.NCAIMRequestProxy
[29] net.straininfo2.logic.url.ForwardRequestProxy
[30] net.straininfo2.logic.hdk.HistriDomainKnowledge
[31] net.straininfo2.logic.url.RequestProxyURLGenerator

the si2 codebase, some custom UrlGenerators and RequestProxys were already rendered obsolete by BRC catalog upgrades. Note that both CatalogLinkOut and RequestProxy are part of the middle tier, although the latter handles HTTP requests. This is because the middle tier needs to be able to determine whether a strain number has a corresponding strain browser URL. This is only possible if the full subsystem is available to the business logic (it needs the RequestProxys for generating their URLs). Moving some of the functionality to the webapp module would introduce a dependency of the core module on the webapp module, which is undesirable.

Some of the redirects could be replaced with an 'internal redirect': instead of returning a HTTP response and depending on the client to perform a new request, the framework internally handles the new URL and directly returns the second HTTP response to the client, which eliminates some HTTP round-trips. However, the current approach has a couple of advantages. Both for the 'No BRC catalog entry' page and the RequestProxyAction pages, a response of the StrainInfo web application is necessary. For the case of a straightforward BRC catalog entry URL, the forward makes it harder for crawlers to obtain the URL and forces the use of the StrainInfo RESTful URLs. In addition, this influences how Google distributes the PageRank. The intermediary StrainInfo URL shows the BRC catalog without the strain browser overlay panel. Although StrainInfo strongly promotes the strain browser, this URL is foreseen for the case that external resources only want to link directly to the BRC catalog entry. In case of improper external use, there remains the possibility of adding a referral check to force the clients to view the BRC catalog entry with the overlay panel.

3.2.3 Tools and sitemap modules (presentation tier)

The tools module contains standalone applications that are used to maintain StrainInfo. All applications reside in package net.straininfo2.tools and often are merely front-ends for business logic calls. The most important tools for maintaining StrainInfo are listed here.

ImportXML Tool to copy MCL XML synchronisation files to the database. The MCL file is parsed into individual mcl:Culture elements, which are individually saved as CatalogXMLRecords[32]. Some fields used by the legacy equivalence integration are readily parsed from the XML CLOB and stored in

[32] net.straininfo2.entities.sync.CatalogXMLRecord

the entity. In addition, the complete CLOB is also stored for later processing. All records belonging to one XML file are linked using a `CatalogXML`[33] record. After running this tool, the equivalence integration (which is a legacy stored procedure) needs to be started.

ParseCultureXMLRecords Last step in the MCL XML synchronisation process (after `ImportXML` and the legacy equivalence integration). The XML CLOB of each record (`CatalogXMLRecord`) is parsed, and all MCL fields are stored as `CultureTriples`. Based on the `mcl:cultureLastUpdateDate` of the CLOB, it is determined if an update of the `CultureTriples` is necessary. If so, all existing `CultureTriples` are removed, and are replaced with the parse results of the current CLOB. Each record is parsed and committed to the database individually, setting `dateProcessed` to the current time after successful processing. This allows importing XML files with partial inconsistencies.

Sitemap `GenerateSitemaps`[34] is used to generate the StrainInfo sitemap. This is a list of all URLs in the web application, formatted using the Sitemap 0.9 protocol[35]. Search engine crawlers automatically use these sitemaps to discover new web pages without having to find their URLs by parsing all pages of a web site. In the case of StrainInfo, many passport pages are hard to discover as there are no or only hard to find incoming links to those pages.

Originally, the web application generated sitemaps on demand. However, the aggressive crawling of different search engine robots generated a large load which decreased performance for real users. Therefore, the Struts2 sitemap generation code (actions and results) was moved to a separate `sitemap` module. In addition, a stand-alone tool generates the sitemap files which are hosted as static files. The tool uses the original Struts2 actions and results, and simulates a request as performed by Struts2. This approach was chosen above fully refactoring the code to keep the implementation of this irregularly used maintenance utility time-efficient.

[33] `net.straininfo2.entities.sync.CatalogXML`
[34] `net.straininfo2.tools.GenerateSitemaps`
[35] `http://www.sitemaps.org/`

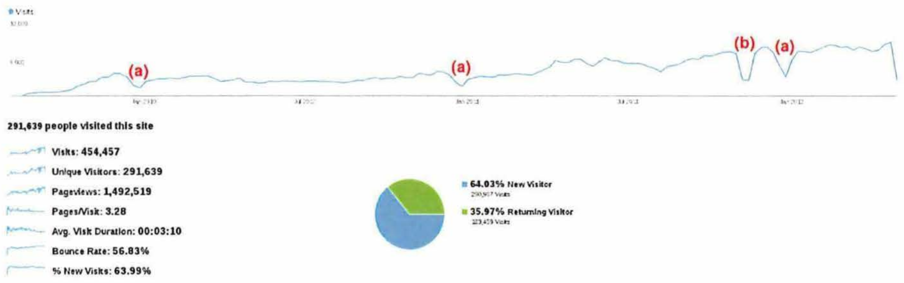
3.3 Daily operation of StrainInfo

The StrainInfo web application can be deployed on a standard J2EE web container such as Apache Tomcat[36] or the JBoss Application Server[37]. In addition, the webapp module also contains a built-in Jetty[38] web server, which allows to run the web application on any machine which has the right Java and Maven versions installed and has access to the Oracle database.

Although the majority of this chapter describes the development effort that has been put into StrainInfo, an important and undervalued part of StrainInfo is its daily operation. In practice, making and keeping StrainInfo available is more than merely deploying the web application to a public web server. This boils down to many different aspects. First, running and maintaining a high performance database and web server, along with supporting infrastructure such as the MCL upload facility, automatic synchronisation with external resources, backup and the development environment (code versioning, continuous integration, etc.) requires a considerable amount of system administration. Unfortunately, hardware does fail, software becomes outdated or even unsupported and security vulnerabilities become discovered. Nevertheless, users expect their web services to be operational at any time. Publishing about StrainInfo means entering into an implicit contract to maintain StrainInfo on the long term.

Second, next to the technical aspects, there is the aspect of maintaining the content, i.e. the StrainInfo index itself. StrainInfo is more than code alone: it is the result of the large integration and curation effort that has been taking place since the conception of the first StrainInfo prototype. Since then, StrainInfo has been gradually acquiring and integrating new BRC catalogs. While initially screen scraping was performed, this was phased out in favour of MCL XML synchronisation, which is less error-prone and has less overhead. However, the effort for keeping the index up to date therefore shifted from writing custom parsers, to providing BRCs support and feedback to generate XML files. Next to integrating BRC catalogs, the curation of the index is essential to not fall into a cumulative cascade of inconsistencies. While this initially was performed by looking at raw database records and trying to discover inconsistencies at sight, a better solution was quickly conceived (see [37]). Later followed Histri (see Chapter 4), which adds a visual historical overview, and enables a new form of curation. The increasing user base

[36]http://tomcat.apache.org/
[37]http://www.jboss.org/jbossas
[38]http://jetty.codehaus.org/jetty/


291,639 people visited this site

Visits: 454,457

Unique Visitors: 291,639

Pageviews: 1,492,519

Pages/Visit: 3.28

Avg. Visit Duration: 00:03:10

Bounce Rate: 56.83%

% New Visits: 63.99%

64.03% New Visitor

35.97% Returning Visitor

Figure 3.5: Google Analytics: weekly visits since the launch of the si2 codebase (August 1st, 2009 – May 1st, 2012). A visit is a use of the StrainInfo web application, and can consist of browsing multiple pages. Note the recurring decrease in visits during the Christmas holiday (marked with (a)). There is no measurement for a period of ten days in November 2011 due to a deployment that did not contain the Google Analytics code (marked with (b)).

also increasingly generates feedback, based on the content of strain passports and Histri. Integration and curation work is essential to keep StrainInfo relevant, but is hard to scientifically recuperate as it is impossible to write papers about routine maintenance work.

Third, there is also the aspect of outreach. This includes promoting StrainInfo world-wide, but also communication with end-users. Since the launch of the si2 codebase, the user base has grown dramatically (see Figure 3.5). As a result, the number of questions coming from the user base has increased in proportion, and these users need to be given a suitable reply. This is important to follow up: questions or suggestions can lead to interesting collaborations (e.g. data sharing or requests for specific functionality), but there is also a burden of questions on properties of certain strains, or even purchase requests.

Fourth, maintaining StrainInfo on the long term means access to funding to support the daily operation. Many funding agencies give preference to starting new work, as opposed to maintaining existing systems. There are funding opportunities for existing infrastructure and projects, but these are scarce and limited in time. Therefore, the strategy for maintaining StrainInfo on the long term is to continue to develop StrainInfo through independently funded case studies. The daily operation of StrainInfo is then seen as overhead in the context of those case studies.

Part II

Improving the integration confidence

4

Histri: History in StrainInfo

This chapter describes how exchange histories of microbial strains can be reconstructed using a semi-automatic curation process and is organized as follows. Section 4.1 shows how to represent and reconstruct exchange histories, and explains their importance. In particular, Section 4.1.3 describes an initiative we have launched to reconstruct exchange histories of bacterial and archaeal type strains from historical information using a manual curation process. Section 4.2 describes our experience with this curation initiative, how exchange histories can be used to track microbial authenticity and meta-information and how this relates to currently existing initiatives. The chapter ends with some conclusions in Section 4.3.

4.1 The exchange history of microbial strains

The exchange history of a strain can be visualized as a rooted tree. An example of the exchange history of the type strain of *Chryseobacterium balustinum* [61–64] is shown in Figure 4.1. This graphical representation of the exchange history is called a 'Histri' in StrainInfo terminology; the name originates from 'History in StrainInfo'.

Cultures are represented as blocks containing their corresponding strain number. The first known culture is displayed at the top of the tree and is marked by a star if it corresponds to the original isolate of the strain. In most cases, the original isolate is assigned a strain number by its isolator. However, in Figure 4.1, the original isolate is labeled with the isolator's name (`C. Tysset`) as no strain number from the isolator is known. Each transfer of a culture between BRCs or researchers is visualized as a line between the originating and resulting cultures. In this particular example, the isolate was subsequently transferred to V. Bonifas and the National Collection of Type Cultures (NCTC, United Kingdom), and upon receipt by NCTC was assigned a strain numer using `NCTC` as an acronym. NCTC subsequently transferred subcultures to the American Type Culture Collection (ATCC), the Collection of Institut Pasteur (CIP, France), the BCCM/LMG Bacteria Collection (Laboratory of Microbiology at Ghent University, Belgium) and the National Collection of Industrial Bacteria (NCIMB, United Kingdom), which, on their turn, further distributed the strain. As a consequence of spelling variants or typing errors, multiple synonymical strain numbers can denote exactly the same culture. These synonymical strain numbers are grouped into the same block. In the example, the first descendant of the original isolate is known under four synonymical strain numbers.

4.1.1 Relevance of exchange histories

A Histri helps in safeguarding the strain's authenticity. If, for instance, in the process of transferring a culture a contamination is introduced, all subsequent cultures (as well as scientific results deduced from these cultures) can be void. This is even more problematic if the contamination is discovered after a long period of time, as in this case such cultures might have been used several times in different research projects. Therefore, Histris are an essential tool in validating the authenticity and correctness of the microbial material. Moreover, a completed Histri allows for easy detection of a wide range of errors. If a strain traces back to two separate lineages (with no trace of a common ancestor), the strain may be the result of a mix-up of two separate strains [37]. The Histri of a strain provides a simple graphical representation and, when interpreted in combination with the StrainInfo catalog cross-reference table [37], mixed strains can be easily discovered and corrected. Evidently, mixed strains can lead to misleading scientific results and should be resolved as early as possible.

Histris could also serve as a cornerstone in building a Material Transfer Agreement (MTA) tracking system. For example, to attribute intermediaries for benefit

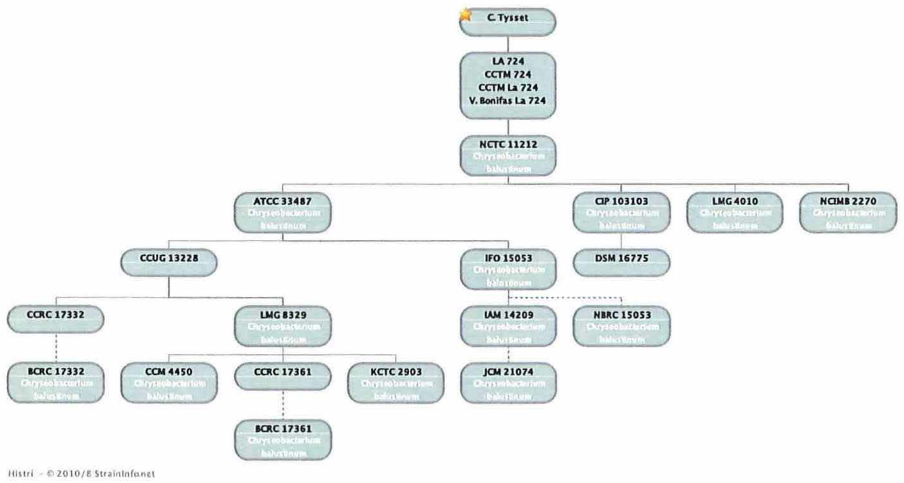

C. Tysset
LA 724
CCTM 724
CCTM La 724
V. Bonifas La 724
NCTC 11212
Chryseobacterium balustinum
ATCC 33487
Chryseobacterium balustinum
CIP 103103
Chryseobacterium balustinum
LMG 4010
Chryseobacterium balustinum
NCIMB 2270
Chryseobacterium balustinum
CCUG 13228
IFO 15053
Chryseobacterium balustinum
DSM 16775
CCRC 17332
LMG 8329
Chryseobacterium balustinum
IAM 14209
Chryseobacterium balustinum
NBRC 15053
Chryseobacterium balustinum
BCRC 17332
Chryseobacterium balustinum
CCM 4450
Chryseobacterium balustinum
CCRC 17361
KCTC 2903
Chryseobacterium balustinum
JCM 21074
Chryseobacterium balustinum
BCRC 17361
Chryseobacterium balustinum
Histri - © 2010/8 StrainInfo.net

Figure 4.1: Example Histri of the *Chryseobacterium balustinum* type strain as reconstructed from the history descriptions in Table 4.1. The isolate of C. Tysset (marked with a star) was transferred to the National Collection of Type Cultures (NCTC, United Kingdom) via V. Bonifas. Subsequently, NCTC further distributed the strain to the ATCC, CIP, LMG and NCIMB BRCs. This Histri also contains four institutional transfers, drawn with dashed lines.

sharing or for validating the legal constraints of all intermediary deposits, a uniform system tracking the original isolate and all intermediary BRCs is necessary. This stresses the importance of the formal recognition of the origin of material, and the need for a public repository tracking transfers of microbial material.

4.1.2 Histri reconstruction

In order to build a central repository that contains the Histris of all strains, the current legacy of microbial material needs to be 'curated' to a new universal representation. This means that Histris need to be reconstructed from the exchange information still available today. A lot of information on how strains were exchanged in the past can be found in BRC catalogs and publications, but also in private communications, general background knowledge or even individual researchers' memories. Although it may seem conceptually simple, combining this information is not at all trivial. Just by considering the scale of this reconstruction process (e.g. the large number of strains and the broad nature of the historic information), it immediately becomes apparent that curating this information requires a large effort. Moreover, additional difficulties such as the imprecision or lack of information, historical informal transfers between researchers and even conflicting information make this a difficult or sometimes even impossible task.

Two distinct types of transfers can be distinguished in reconstructed exchange histories. Most Histri transfers distribute the culture to other BRCs or individual researchers and therefore are economic in nature. Hereby, clones of individual cultures are sent physically to other parties, effectively contributing to the global distribution of the microbial material, and hence are named 'distributional transfers'. A BRC that receives the material becomes a node in its further distribution. Most BRCs provide a textual description of the historic pedigree of a culture in their catalogs. Table 4.1 lists all history descriptions extracted from BRC catalogs that were used to reconstruct the Histri in Figure 4.1. Sometimes only the direct predecessing strain number is given, but generally all other intermediate strain numbers up to the original isolate are given. Frequently, the information is not accurate and the use of contextual information can be necessary to determine the correct predecessor, such as in cases where only the acronym of the ancestral BRC is given. When two or more strain numbers with that particular acronym are known for that strain, contextual information such as the lineage, the deposit date or the 'other collection numbers' listed by the BRC must be used to determine the correct predecessor. For example, a number of history descriptions in Table 4.1 list `LMG` as the immediate

Strain number	Original history description
ATCC 33487	ATCC <<--NCTC<<--V. Bonifas La 724 <<--- C. Tysset
BCRC 17332	<< CCUG << ATCC<< NCTC << V. Bonifas La 724 << C. Tysset
BCRC 17361	<< LMG << CCUG (Flavobacterium balustinum)
CCM 4450	< LMG < CCUG
CCUG 13228	n/a
CIP 103103	<-- 1988, NCTC, Flavobacterium balustinum <-- V. Bonifas, Lausanne, Switzerland: strain La 724 <-- 1959, C. Tysset
DSM 16775	<- CIP <- NCTC <- V. Bonifas, Lausanne, Switzerland; La 724 <- C. Tysset.
JCM 21074	<-- IAM 14209 <-- IFO 15053 <-- ATCC 33487 <-- NCTC 11212 <-- V. Bonifas LA 724 <-- C. Tysset.
NBRC 15053	IFO 15053 <- ATCC 33487 <- NCTC 11212 <- V. Bonifas <- C. Tysset
KCTC 2903	<- LMG <- CCUG
LMG 4010	n/a; after personal communication: <- 1980, NCTC <- V.Bonifas <- 1959, C.Tysset
LMG 8329	<- 1988, CCUG (Flavobacterium balustinum)
NCIMB 2270	Depositor Company: National Collection of Type Cultures (NCTC); History: V.Bonifas -- C.Tysset
NCTC 11212	Depositor: BONIFAS V; History: BONIFAS V, CETRE DE COLLECTIONS, LAUSANNE-ISLT BY TYSSET C PRE:FR

Table 4.1: Original history descriptions of the strain depicted in Figure 4.1 as given by the different BRCs. This is the literal, unformatted history information as found in the corresponding online catalog entries. The catalog entries for CCUG 13228 and LMG 4010 are no longer available. The history description of the latter was recovered from the StrainInfo cache and confirmed by personal communications.

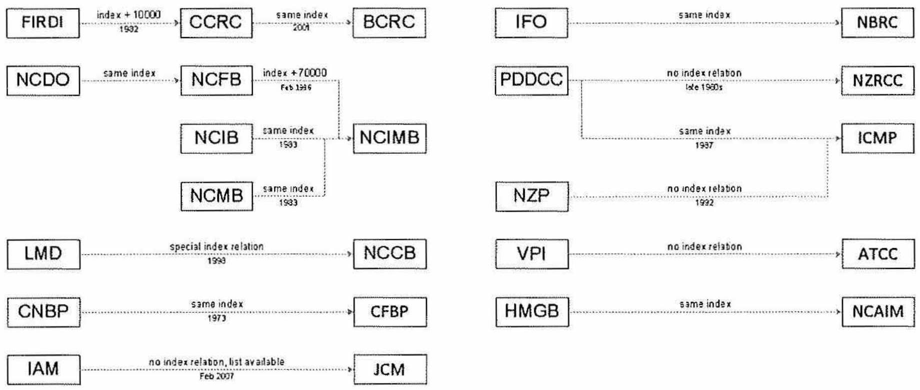


Figure 4.2: Overview of all institutional changes used by the Histri inference engine and their relation to strain numbers.

predecessor. As there are two LMG strain numbers known for the strain, information about the lineage (i.e. the LMG originating from CCUG) must be used to decide upon the correct ancestor. Inconsistencies also occur, e.g. descriptions where the order of the predecessors has been changed (this error could have been introduced by the collection curator incorrectly entering data into the catalog database). As a consequence, the textual history descriptions are not well suited for automatic processing. This type of information is sometimes also available in publications.

Distributional transfers stand apart from 'institutional transfers'. These transfers can be conceived as being virtual and originate solely from institutional changes, such as two BRCs merging or a BRC changing its name. Although the material is often not physically relocated, this sometimes can be the case. Typically a relatively large number of strains are affected, but this background knowledge only needs to be acquired once. An overview of these transfers can be found in Figure 4.2. Although the World Data Center on Microorganisms (WDCM) of the World Federation of Culture Collections (WFCC) [39] maintains a list of recognized BRCs and the *List of Prokaryotic names with Standing in Nomenclature* of Jean Euzéby [26] has a section listing BRC acronyms together with some additional contextual information, there is no central repository keeping track of these institutional changes. A list of acronym changes based on the list of Euzéby and additional information discovered in catalogs, on websites and by personal communication has been created. Most institutional transfers only result in the replacement of the acronym, while the index is kept the same. However, the transfers from LMD (Laboratory of Microbiology, Delft, The Netherlands) to NCCB (Netherlands Culture Collection of Bacteria) and from NCFB (National Collection of Food Bacteria, UK) to NCIMB (National Collections of Industrial Food and Marine Bacteria, UK) have a special (but algorithmically defined one-to-one) relation between the indices. Additionally, JCM (Japan Collection of Microorganisms) has made a list available of the transfer between IAM (Institute of Applied Microbiology, University of Tokyo, Japan) and JCM. In completed Histris, this type of transfer is displayed using dashed lines, whereas transfers of the first type are represented using solid lines (the example in Figure 4.1 contains four institutional transfers).

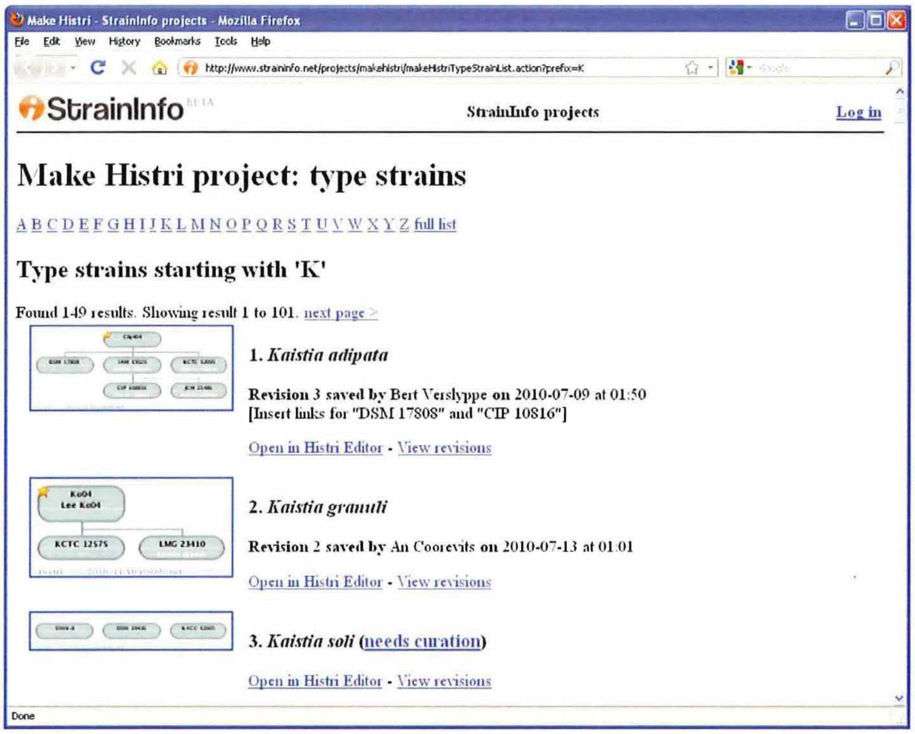
Certain acronyms are used interchangeably and can be regarded as synonyms, as there is no clear historic distinction between them. Mostly this originates from a confusion between the collection and the institution acronym. Acronyms regarded as synonyms by StrainInfo are `AS` = `CGMCC`, `BKM` = `VKM`, `CDC` = `NCDC` and `DSM` = `DSMZ`.

4.1.3 The 'Make Histri' project: manual curation of bacterial and archaeal type strains

As a case study for the complete curation of all microbial strains, our goal was to reconstruct the Histris of all bacterial and archaeal type strains. Due to their specific role in bacterial and archaeal taxonomy (each species has a designated type strain), type strains are generally more broadly utilized in research. As a consequence, they have a broader global distribution, often have more detailed catalog entries and have better coverage in literature. These factors make them an interesting and well-defined subset of all microbial strains. Although this subset may seem relatively small compared to the total number of strains in StrainInfo (4.3%), the large absolute number of type strains (8998 as of August 1^{st}, 2010) makes it clear that multiple curators are needed. Therefore, a wiki-like community annotation process was envisioned.

As a billboard for the community of curators, a project home page was created[1]. The home page is the central entry point for people involved in the annotation process. The most important component of the home page is the list of type strains as shown in Figure 4.3. Due to the large number of species, the list is split into separate pages. These pages list all bacterial and archaeal species names,

[1]`http://www.straininfo.net/projects/makehistri/`



Make Histri - StrainInfo projects - Mozilla Firefox

http://www.straininfo.net/projects/makehistri/makeHistriTypeStrainList.action?prefix=K

StrainInfo projects

Log in

Make Histri project: type strains

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z full list

Type strains starting with 'K'

Found 149 results. Showing result 1 to 101. next page >

1. *Kaistia adipata*

Revision 3 saved by Bert Verslyppe on 2010-07-09 at 01:50
[Insert links for "DSM 17808" and "CIP 10816"]

Open in Histri Editor - View revisions

2. *Kaistia granuli*

Revision 2 saved by An Coorevits on 2010-07-13 at 01:01

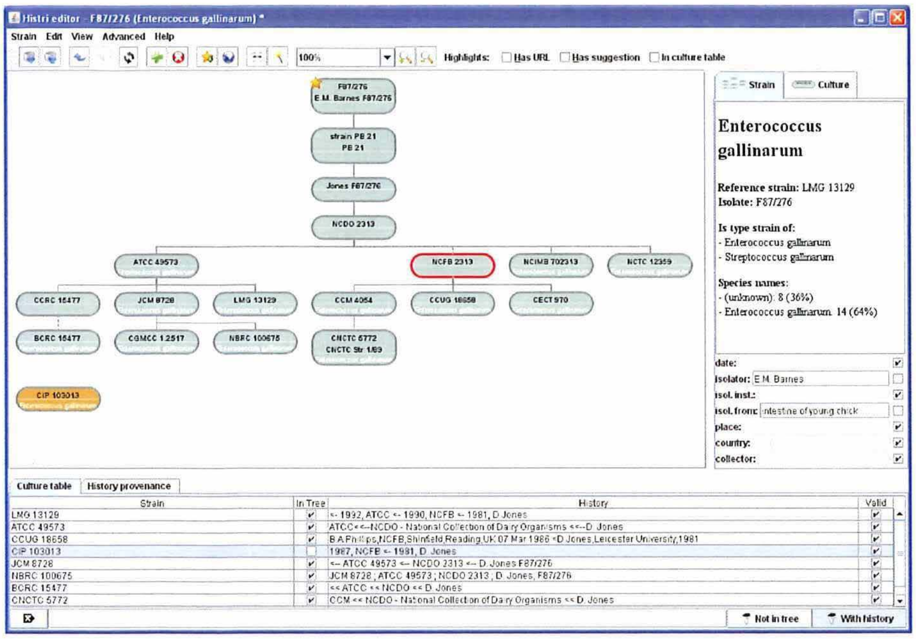
Open in Histri Editor - View revisions

3. *Kaistia soli* (needs curation)

Open in Histri Editor - View revisions

Done

Figure 4.3: Type strain list of the 'Make Histri' project. On this page, the first two strains have been reconstructed, while the third still needs manual curation. Users can save multiple revisions and add a comment while saving.



Enterococcus gallinarum

Reference strain: LMG 13129
Isolate: F87/276

Is type strain of:
- Enterococcus gallinarum
- Streptococcus gallinarum

Species names:
- (unknown): 8 (36%)
- Enterococcus gallinarum: 14 (64%)

date:		✓
isolator:	E.M. Barnes	☐
isol. inst.:		✓
isol. from:	intestine of young chick	☐
place:		✓
country:		✓
collector:		✓

Culture table | History provenance

Strain	In Tree	History	Valid
LMG 13129	✓	<- 1992, ATCC <- 1990, NCFB <- 1981, D Jones	✓
ATCC 49573	✓	ATCC<<--NCDO - National Collection of Dairy Organisms <<--D. Jones	✓
CCUG 18658	✓	B.A.Phillips,NCFB,Shinfield,Reading,UK 07 Mar 1986 <D Jones,Leicester University,1981	✓
CIP 103013	☐	1987, NCFB <- 1981, D. Jones	✓
JCM 8728	✓	<-- ATCC 49573 <-- NCDO 2313 <-- D. Jones F87/276	✓
NBRC 100675	✓	JCM 8728 ; ATCC 49573 ; NCDO 2313 ; D. Jones, F87/276	✓
BCRC 15477	✓	<< ATCC << NCDO << D. Jones	✓
CNCTC 6772	✓	CCM << NCDO - National Collection of Dairy Organisms << D. Jones	✓

Figure 4.4: View of the Histri editor while reconstructing the exchange history of a strain. General information on the strain (including isolation date, isolator and affiliation, habitat description of the sample and corresponding location) can be added on the `strain tab` (side panel). The suggestion system suggests a possible ancestor (red border) of the currently selected strain number (orange).

i.e. all strains needing curation in the case study. Completed Histris stay on the list to show the project progress. Strains that need curation are clearly marked as such and users can edit the Histri of the corresponding type strain using the Histri editor. The Histri editor runs on the Java 1.6 platform and is automatically launched using Java Web Start technology. The editor immediately loads the strain, and as users need to be authenticated to save a Histri, the user's session is passed on to the editor (see Figure 4.4).

Histris are built using drag and drop. To insert a transfer, a culture is dragged and dropped onto its ancestor. This creates a link between both cultures that can be annotated with additional information such as the deposit date or the depositor's name by entering the information in the fields on the culture tab (side panel of the

Histri editor). Users insert links corresponding to the history information as found in the corresponding BRC catalog entries. To impose no hurdle on the user, the editor includes direct links to the relevant catalog entries through the StrainInfo strain browser.

We recruited about thirty researchers as biocurators and launched the project with two training sessions. During these sessions, the aim of the project was explained and users were instructed how to reconstruct Histris using the software. At the end of the session, the attendees were given a hands-on assignment to curate their first strains. Under the motto 'Make Histri instead of playing Patience', we asked the people to curate a few strains whenever they had free time, for example while waiting during experiments or after working hours. After the initial launch, we observed a steady increase in the number of completed Histris for a few months, after which participation ran down and the growth diminished. At this point more than 4000 type strains were curated in a period of five months. This corresponds to about 400 man hours of voluntary work.

In order to complete the goal of curating all bacterial and archaeal type strains, a change of course was necessary. During the project, the Histri editor was further improved on the basis of feedback from users to make the curation of strains as smooth as possible. To aid users in finding the right predecessor, a suggestion system was introduced. This system suggests possible predecessors based on built-in history descriptions by marking those blocks with a red border, making it easier to be distinguished by the user. On top of this, an autobuild feature automatically inserting transfers with a high certainty was developed. Although initially conceived as an experimental aid, it proved to be more effective than originally expected. As more manually curated Histris became available, the algorithm used by the autobuild feature was further improved by comparing the output of the algorithm to the manually annotated strains. The error rate was reduced to less than 2%, with a miss rate[2] of about 5.5%. Some of the discrepancies between the autobuild algorithm and the human results were identified as human errors and obvious errors were corrected. This suggests that the effective error rate is lower. However, it was observed that the correctness of the algorithm decreases as the number of cultures in the strain increases, while strains with few cultures (i.e. less than 10 strain numbers) are built virtually perfectly. Therefore, human effort was concentrated on complex, historically important, strains while other strains were built by autobuild, without

[2]The fraction of transfers found by users, but not by the algorithm. This is missing information which can be added later when new information becomes available. Therefore, this is not regarded as erroneous.

manual intervention. These strains with many cultures take considerable effort and time, and as a consequence, this significantly decreased the users' readiness to contribute to the project, due to their preference for strains with few cultures that are easier to build and require less time. By automating the easy Histris that do not need human judgement, the curating community became less active and evolved into a community of users curating strains relevant to their personal research, but not necessarily part of the 'Make Histri' project.

4.2 Discussion

We experienced that the academic community is open to do voluntary work for the 'greater good' without direct personal benefits. Nevertheless, the community lacks the mass reached by popular services such as Wikipedia. As a consequence, even with good visibility, the expected feedback or input received from users is of the same magnitude. This does not imply that small curation projects are impossible: in fact, we experienced that the scientific community is willing to help if people are personally involved and the effort required is minimal. However, large or difficult curation and annotation projects are only possible if people have clear incentives to do so (payment, official requirement, etc.).

As we are shifting towards an era where scientists increasingly make use of publicly available data rather than having to generate their own data, the quality of the data and their corresponding metadata becomes increasingly important. Unfortunately, information in large public databases is often difficult to process automatically, of mediocre quality, is hardly annotated or out of date. This is true for the majority of material in online repositories: data are created and deposited, but never updated again. To be able to successfully make use of this legacy, data need to be converted to meet the proper digital and quality standards. Whether we are able to clean legacy data will depend on the priorities of funding agencies, but due to the high cost and effort this is unlikely to happen. We therefore need to seek ways to capture this meta-information in electronically processable ways, making it available for the future.

4.2.1 Histri as a framework for long-term tracking of microbial authenticity and meta-information

The meta-information problem is also apparent in the preservation of microbial material. As discussed in Chapter 1, microbial material is deposited in BRCs where, after quality control, it is assigned a strain number, and together with additional meta-information, is entered in the BRC catalog to be able to retrieve the culture. The culture itself is preserved using special techniques such as lyophilization, making it possible to revive it at a later stage. This allows the material to be transferred to other BRCs and researchers, and as a consequence, the material can become globally distributed. Although the biological material, and thus the corresponding meta-information about the culture, is supposed to be equivalent and complementary, most actors maintain their own local information system. Consequently, next to the material itself, the corresponding meta-information also becomes globally distributed as it is maintained (or in practice rather archived) by the different actors. In order to keep their material linked to equivalent cultures of the strain, BRCs list equivalent strain numbers known at the time of deposit in their catalogs. Based on the redundant information in these lists, the StrainInfo integration engine is able to successfully integrate the different strain numbers into strains, and to apply error detection and error correction (see Chapter 2). By having all equivalent strain numbers, it becomes possible to link to much more downstream information (such as sequences or publications linked using equivalent strain numbers) available for the strain.

Unfortunately, this does not guarantee the authenticity of the referenced material, and thus the legitimacy of using results obtained from equivalent cultures. In some cases, equivalent strain numbers are mixed up, which results in strain numbers of different strains being regarded equivalent while they are not. Although the StrainInfo integration algorithm was built to detect such cases, it will fail to detect situations where the errors have percolated in a large fraction of the catalogs of BRCs where a culture of the strain is being held. In this case, however, it is often possible to detect the confusion by reconstructing the exchange history of the strain. If the strain traces back to two or more lineages that cannot be linked through a common ancestor, we are often facing a strain that is a merger of multiple strains, and this strain will need human curation (e.g. be split into two separate strains).

Most BRCs list both the depositor strain number and the complete exchange history in their catalogs. In theory, only the former is needed to be able to reconstruct the Histri of strains as it is the minimal information required to link a culture to its predecessor, and it is also relatively easy to process electronically. However, in order to successfully reconstruct Histris, this information should be maintained very carefully as a small error in the cited strain number (e.g. a swap of two digits) could link the culture to another strain, effectively introducing a mix-up of two distinct strains. Moreover, as there are ambiguity problems with strain numbers, a proper identifier needs to be used as a substitute when establishing long-term links. Therefore, StrainInfo introduced the cultureId (see Section 2.2.4) which overcomes many of the problems with strain numbers as currently in circulation, but this comes at the cost of resolving between strain numbers and cultureIds. However, if the ancestor cannot be unambiguously linked, the culture should not be linked to a strain. Although this may incur the loss of access to equivalent information, it guarantees that no false assumptions are made. Consequently, despite the fact that the direct predecessor fields theoretically are sufficient to reconstruct the complete Histri, in practice this would lead to incomplete Histris. Classic textual history descriptions therefore stay important as they document preceding exchanges, add additional information and are used in many legacy strains. In cases where no direct predecessor is explicitly given (e.g. cultures not in public collections), the redundancy in the history descriptions can be used to recover the correct origin for all cultures on the path to the isolate, and it assures that the culture can be linked to the isolate.

Even if this information is perfectly maintained in catalogs, it needs to be centralized and integrated to be further processed electronically. This can be done with the Microbial Common Language (MCL), which has provisions for this type of information (see Chapter 6). Two separate mechanisms can be used to describe historic information. First, the `mcl:history` term can be used to describe the exchange history of material using the textual descriptions as they are currently used in BRC catalogs. This is a so-called legacy term, supporting current practices in existing databases. Next, individual transfers can be modelled using the `mcl:Deposit` entity. Each transfer can be described with fine-grained annotations. Moreover, this element is used at both the culture and the strain level. At the culture level, it can be used by BRCs to exchange fine-grained historic information if available (or the direct predecessor as described above), whereas at the strain level, it is used by StrainInfo in its strain exports. Note that MCL is used in two distinct directions, i.e. the collection of historic information and the dissemination of the integrated information. Information enters StrainInfo at the culture level, and is integrated by StrainInfo to make it available at the strain level in a format that may be processed electronically.

Historic information is too important to be reconstructed afterwards. Along with the preservation of the biological material itself, it is equally important to take care of the describing meta-information by requiring to conform to both quality and electronic formatting standards. This can only be enforced at the time the data are created, since once meta-information reaches the catalog it is too late (i.e. it risks of never getting updated again). For maximal adoption, this should be dealt with in a very practical way. By requiring a minimal description of the deposit and the use of MCL to format this information in an electronically processable way, already a big step towards rich and complete Histris is made. Once data become accessible in a machine readable form, this will open the upgrade paths to more sophisticated representations.

4.2.2 Registration of transfers and relation to the International Treaty on Plant Genetic Resources for Food and Agriculture (ITPGRFA)

In an ideal world, the reconstruction needs to be performed only once because for new transfers an obligatory registration system could be enforced to guarantee the archiving of historic information. Transfers could be registered using web services at the time the transfer is initiated, and confirmed upon verified receipt. Such a system is already in place in the context of the International Treaty on Plant Genetic Resources for Food and Agriculture (ITPGRFA) [65, 66]. The ITPGRFA aims at conserving and making sustainable use of Plant Genetic Resources for Food and Agriculture (PGRFA) and at a fair and equitable sharing of benefits derived from their use. The Multilateral System for Access and Benefit Sharing (MLS) is based on a Standard Material Transfer Agreement (SMTA) that accompanies any transfer of PGRFA under the MLS. One of the requirements of the MLS is reporting transfers to the ITPGRFA secretariat. After the system came into function in 2004, the secretariat reported receiving an enormous amount of data in different and incompatible formats, and that manual storage of the data is impracticable. As electronic processing and storing of such data is the most efficient and cost-effective solution, work was started on an electronic system that consists of a Persistent Identifier (PID) Server that uniquely assigns and manages PIDs to providers and recipients of PGRFA under the MLS, and a Provider Ordering Toolkit (POTK), that handles all SMTA-related activities, including sending SMTA reports to the Governing Body of the ITPGRFA. These reports can be sent real-time (using a powerful filtering interface allowing to select which information is sent to the datastore) or in batches according to a configurable schedule.

While there is ongoing work on standardizing Material Transfer Agreements in for example the European Culture Collections' Organisation (ECCO), there is no universally used MTA between microbiological BRCs. Therefore, MTAs used in distributional transfers are currently ignored in Histris, but the system could be easily expanded to discriminate between different MTAs if this would become necessary. A standardized MTA (with optional clauses declaring additional rights or limitations) would allow to reason with MTAs and automatically determine the legal rights and status of particular material. However, the legal status of material as it is currently distributed needs to be resolved, and this is, due to the historic distribution of strains, a difficult task.

4.2.3 Global distribution of type strains

Histris give insight in how BRCs collaborate to facilitate world-wide distribution of type material. Figure 4.5 shows the aggregated transfer counts of type strain exchanges between BRCs. Each arc represents the transfer of strains between two different BRCs, whereby the stroke thickness corresponds to the number of strains. The bar chart shows the total count of transfers per BRC; the blue bars (left side) correspond to inbound transfers, and the yellow bars (right side) correspond to outbound transfers. Note that this figure does not take non-type strains into account, nor transfers to researchers or industry, nor collection size. Autobuild was used to reconstruct strains needing manual curation.

Most new type strain isolates are transferred to well-known BRCs such as the German Collection of Microorganisms and Cell Cultures (DSMZ), the American Type Culture Collection (ATCC), the Japan Collection of Microorganisms (JCM), the BCCM/LMG Bacteria Collection (Laboratory of Microbiology, Ghent University, Belgium), the Collection of Institut Pasteur (CIP, France) and Culture Collection University of Göteborg (CCUG, Sweden), with a preference for DSMZ and ATCC. Next, large BRCs collect rather than provide type strains to other BRCs. This can be seen by those BRCs having more inbound transfers than outbound transfers. Notable exceptions include ATCC and NCIMB (United Kingdom), which have more outbound transfers than inbound transfers. It is also apparent that most BRCs have preferences in their suppliers of additional material originally deposited elsewhere and generally will order additional material from a handful of accustomed BRCs. In some cases, some BRCs have very strong bonds with other BRCs: for example, most (74.6%) strains of VTT (Finland) were deposited by DSMZ.

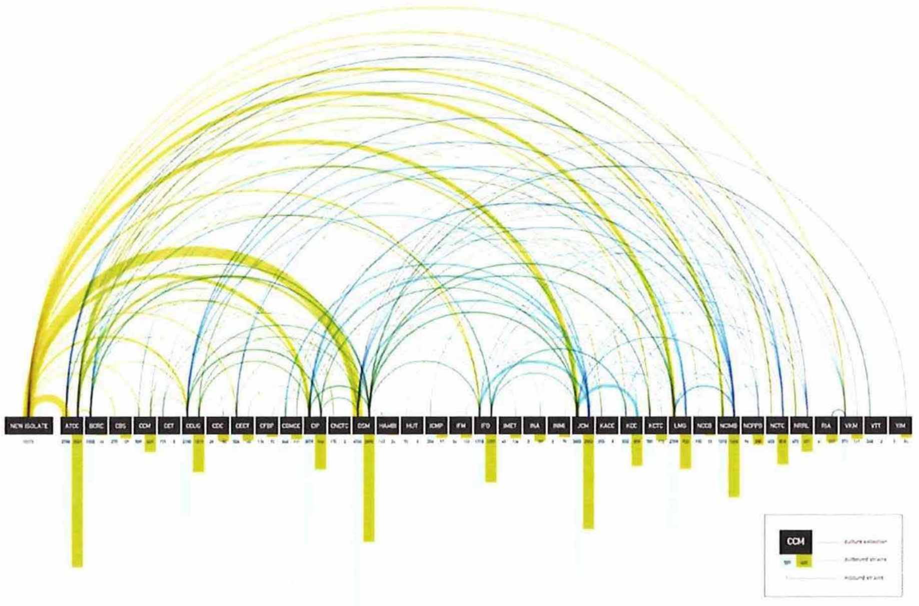


Figure 4.5: Distributional transfers between BRCs. Each arc represents the transfer of strains between BRCs, whereby stroke thickness corresponds to number of strains. The total inbound and outbound transfer count is visualized in the bar chart. For clarity, arcs representing transfers of less than 30 strains were omitted (this does not affect the bar chart).

4.2.4 Granularity of Histris and the availability of material

Histris model transfers between different BRCs, but it is also possible to use Histris to model at a more fine-grained level, for example by incorporating internal BRC lots or batches. This is clearly useful, as for instance illustrated in [67] where the authors unravel differences in the type strain of *Mesorhizobium loti* as held by different BRCs. It is suggested that a distribution lot of the ATCC 33669 culture exhibits two colony types: the first colony type corresponds to the material as available in most other collections and matching most of the sequences historically available from the INSDC repositories, while the second colony type shows enough differences to even belong to another species.

This could be taken further to the point where each subculturing step could be seen as a node in the Histri of the strain. This would allow one to trace back authenticity problems by pinpointing their precise origin on the bench, but this comes at the cost of an extensive administration. As a side-effect, this would also yield Histris with a large number of relatively short-lived nodes that quickly become unavailable as after subculturing the original cultured plates are discarded. BRCs often employ special techniques to allow long-term preservation of material, making their cultures last for a longer time period. As a result, there can be a large time difference between the first and most recent branch created by a BRC. At the same time, current Histris already contain a large number of unavailable nodes. This is the case if the original isolator's cultures are listed in the Histri (original plates discarded after deposit) or when a culture held at a BRC looses its viability. In the latter case, it may happen that a BRC silently recovers its collection by obtaining a new culture of the same strain from another BRC. However, it should be a formal requirement that a redeposit results in the allocation of a new strain number, even if the material is obtained from the same provider. This makes sure that the redeposit is recognized as such, and that the Histri will give a correct account of the historical situation. This is illustrated by the Histri of the *Geobacillus stearothermophilus* type strain containing the strain number CECT 43 twice. In this case, the catalog clearly mentions the reaccession and lists the original history, but as the original deposit is in a historically distant branch, there is an increased risk of authenticity problems.

4.2.5 Histri editor design considerations

The Histri editor has been developed as a separate stand-alone tool and uses web services to communicate with StrainInfo. Although modern web technologies (such as Adobe Flash) allow the building of tools of similar functionality, the editor has been implemented as a stand-alone Java application in view of its performance, scalability and its large collection of pre-existing open source libraries. Having access to a full-fledged programming language allows the inclusion of more advanced algorithms and eases the reuse or incorporation in other tools.

It has been shown that by modelling curation problems as computer games, large communities that generate meta-information as a side product of their game play can emerge [68, 69]. Although our approach has been influenced by this work, we did not try to exploit the full potential of creating a competitive environment. In such environment, additional mechanisms that guarantee the accuracy of outputs generated by users need to be added to the system. Users are creative in finding ways to gain a higher score with less effort, circumventing the system by performing unintended actions. For this particular task, it is hard to avoid the inadequate curation of Histris without sacrificing curation coverage in terms of number of fully curated distinct strains. Therefore, we designed the 'Make Histri' project website as a manual curation initiative aimed at a specialized target user group (as opposed to a general online community). Nonetheless, the Histri editor was designed to be as smooth as possible so as not to impede the user by avoiding superfluous manipulations. We tried to put the focus on what is hard for computers, but easy for humans. For example, the editor uses animation to help users to keep track of the tree as its layout changes after an insertion, and there is intelligent automated switching of the `culture` and the `strain` tabs. An important assistive feature is the suggestion system which visually aids the user in finding the correct ancestor by displaying suggested ancestors with a red border (see Figure 4.4). Moreover, the tool also includes a `history provenance` tab which helps analyzing the built-in history descriptions. If this information has been indexed by StrainInfo, it is listed in the editor and is used by the suggestion system. The `culture table` also displays the built-in history descriptions and its as-you-type filter can be used to quickly locate particular cultures.

A provenance system keeps track of all information sources and user actions. This information can be viewed by expert users to identify the grounds of suggested transfers, and can also be used to distinguish between automatically inserted transfers and human-made links. The provenance system records the time used to reconstruct the Histri to be able to calculate the total curation time. StrainInfo stores all provenance for future reference.

In addition to the overview from the 'Make Histri' project homepage, Histris can also be edited directly from their corresponding strain passport by clicking the `edit`

Histri button. Using this mechanism, users can curate the strains they need (not limited to bacterial and archaeal type strains) and therefore, *ad hoc* curation of Histris will supersede Histri curation through the Make 'Histri project' homepage. Basic access functionality has been included on strain passports to avoid the need to log in or load the full-fledged editor to view the corresponding Histri. Each strain passport displays its corresponding Histri and the user can view all extra annotations in the web browser using the `show Histri annotations` feature. By hovering over the cultures, the annotations appear in a small pop-up panel. Strains not yet manually curated are built using autobuild. Old revisions and associated meta-information can also be consulted, and the Histri can be exported to PDF and PNG formats for embedding in other resources. The editor requires users to save the Histri before exporting to prevent publication of Histris that have been tampered with. Upon saving, a range of sanity checks is performed to ensure the user does not save erroneous or incomplete trees. To avoid vandalism, and as a way to credit users for their work, logging in is necessary. This also encourages users to save Histris with a higher accuracy and improved completeness.

4.3 Conclusions

Histris give an integrated overview of the exchange history of strains by integrating historic information captured in BRC catalogs and other resources. Historic information held in BRC catalogs only locally describes the history of the cultures. By integrating all descriptions, a complete overview of the strain can be obtained. Users can view Histris on StrainInfo strain passport pages and conveniently edit Histris with the Histri editor. This allows users to detect and cope with situations where strains possibly have authenticity problems.

Integrated electronic access to microbiological information is becoming increasingly important in the emerging field of computational microbiology. Histris guarantee that information corresponding to validated authentic equivalent material can be safely used in large-scale electronic processing. As inauthentic material can possibly have a large impact on obtained results, the stronger authenticity that is established, the higher the confidence level of inferred downstream results. Therefore, Histris assure that scientific results obtained today will be safely usable in the future and, as a consequence, they protect past, current and future microbiological research. The enormous potential of future computational research combined with possibly increased reconstruction problems due to the loss of historic records outweighs the relatively small effort of curating Histris now and installing a proactive system to maintain Histris at the moment they originate. Historic information not being available or even lost in the future makes it then impossible to link cultures to their Histri, and therefore also incurs a possible loss of scientific legacy. Therefore, we invite the microbiological community to actively continue the protection of this legacy by further curating strains in the 'Make Histri' project or other strains relevant to research, but not in the original scope.

This chapter is based on:

B. Verslyppe, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. Make Histri: reconstructing the exchange history of bacterial and archaeal type strains. *Systematic and Applied Microbiology*, 34(5):328–336, 2011.

We would like to thank Tom Dedeurwaerdere for providing information regarding the ITP-GRFA. Gully Burns for his helpful comments on the manuscript of the publication that forms the basis for this chapter. Figure 4.5 was designed by Form Apparatus. We are grateful to Claudine Vereecke and the BCCM/LMG collection for their valuable input from the BRC perspective and their help in resolving the history of the LMG 4090 culture. Furthermore, we want to thank the numerous volunteers from the Laboratory of Microbiology as well as international researchers for their valuable curation efforts in the Make Histri project.

5 Histri editor software architecture

This chapter documents some of the architectural ideas and software engineering solutions used to develop the Histri editor. The Histri editor is written as a Java Swing application and is started from the StrainInfo web application using Java Web Start. The code makes extensive use of Java 1.5 generics, but depends on Java 1.6 because of the use of the `java.awt.Desktop` class, which gives platform independent access to a web browser.

5.1 General design

The application is built using Maven and consists of three modules: `core`, `private-func` and `analysis`. After gaining experience with using Maven in the si2 codebase, Maven was added later to the Histri codebase. Maven makes Histri more modular, making it easier to maintain and build the code. In addition, it makes it possible to transparently include it as a dependency in the si2 codebase (the export to PNG functionality is used to deliver rendered Histri PNG images to the web browser).

The `core` module contains the Java Web Start application as it is made available online. It consists of the application data model (which belongs to the package net.straininfo.histri.model) and the GUI (package net.straininfo .histri.gui). The data model revolves around base class HistriModel[1]. GUI elements consist of the Histri editor window, the HistriPanel[2], the side panels, other widgets and all user Actions[3] (see Figure 5.1). Classes containing functionality not intended for end-users are held in the privatefunc module. The latter contains advanced Actions not used in the public GUI and speciality classes for loading the Histri data from different data sources. The analysis module contains (as expected) stand-alone tools for performing analysis of Histri data. It contains the Bulkfeeder[4] class, providing a framework for operations (implementations of the Consumer[5] class) that need to be executed for a large number of strains.

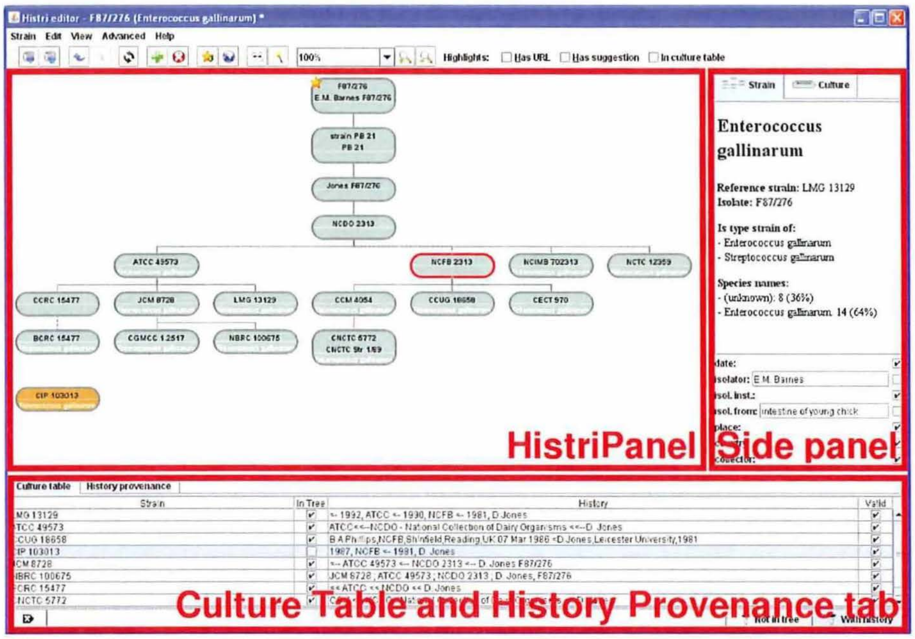
5.2 Application model

At the core of the application is its model: an abstract representation of the application's data. Using the Model View Controller (MVC) design pattern, changes to the model are reflected in the GUI. Views listen to models by registering themselves as listeners. After an update (that effectively updates the model), the model fires an event, giving all listeners the opportunity to react on the change. This is for example used to update the GUI when the Histri tree changes or after a selection event. All changes to a model result in firing events. Therefore, user Actions only perform modifications to the model, which automatically results in an update (repaint) of dependent GUI components.

5.2.1 Core model

The HistriModel is the core model of the application. It maintains the Histri tree and has links to submodels, specialized in more or less separate functionality. Figure 5.2 contains an overview of the submodels (discussed in Sections 5.2.2 and 5.2.3) and their relation to the HistriModel. Having separate models for independent features allows greater granularity of events. This means that instead of generating one single event for all different types of model changes, the (sub)model will fire a

[1]net.straininfo.histri.model.HistriModel
[2]net.straininfo.histri.gui.histripanel.HistriPanel
[3]javax.swing.Action
[4]net.straininfo.histri.analysis.tools.Bulkfeeder
[5]net.straininfo.histri.analysis.tools.Consumer



Enterococcus gallinarum

Reference strain: LMG 13129
Isolate: F87/276

Is type strain of:
- Enterococcus gallinarum
- Streptococcus gallinarum

Species names:
- (unknown): 8 (36%)
- Enterococcus gallinarum 14 (64%)

date: ✓
isolator: E.M. Barnes
isol. inst.: ✓
isol. from: intestine of young chick
place: ✓
collector: ✓

Culture table | History provenance

Strain	In Tree	History	Valid
MG 13129	✓	<- 1992, ATCC <- 1990, NCFB <- 1981, D Jones	✓
TCC 49573	✓	ATCC<<--NCDO - National Collection of Dairy Organisms <<--D. Jones	✓
CUG 18658	✓	B A Phillips,NCFB,Shinfield,Reading,UK 07 Mar 1986 <D Jones,Leicester University,1981	✓
IP 103013		1987, NCFB <- 1981, D. Jones	✓
CM 8728	✓	<-- ATCC 49573 <-- NCDO 2313 <-- D. Jones F87/276	✓
BRC 100675	✓	JCM 8728 ; ATCC 49573 ; NCDO 2313 ; D. Jones, F87/276	✓
CRC 15477	✓	<< ATCC << NCDO << D. Jones	✓
NCTC 5772	✓	[illegible]	✓

Figure 5.1: GUI elements of the Histri editor.

specific event to specific listeners. This is particularly important for listeners that perform relatively computationally intensive tasks, such as for example recalculating the Histri tree layout. Two submodels (`SuggestionManager` and `UndoManager`) also register themselves as listeners of the `HistriModel` as they react on changes to the Histri tree held by `HistriModel`.

The Histri editor was implemented before the development of the si2 codebase was started. Therefore, the terminology and modelling used by `HistriModel` is slightly different from what is later used in the si2 codebase. Cultures are called *atomic* cultures, and are implemented in a `Culture`[6] class specific to Histri. Synonymous strain numbers are grouped as blocks in the Histri editor and are modelled as groups of cultures using the `CultureGroup`[7] class. The Histri editor is centered around those `CultureGroups`: they are the entities users interact with (dragging

[6] `net.straininfo.histri.model.Culture`
[7] `net.straininfo.histri.model.CultureGroup`

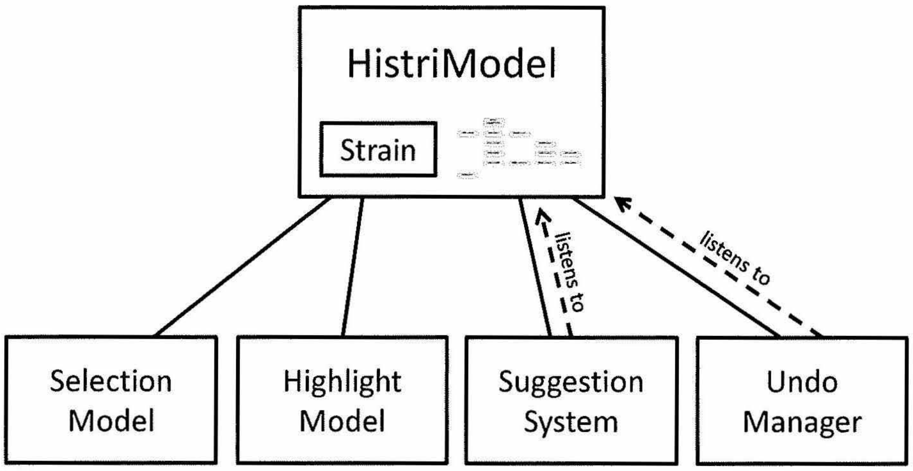


Figure 5.2: All (sub)models of the Histri editor and their relation to the core model `HistriModel`.

and dropping blocks) and the tree structure is applied to them. When a user inserts a transfer, an edge (`CultureEdge`[8]) is added to the model. `CultureEdges` model edges between two `CultureGroups` and contain all metadata associated with them (i.e. all data entered on the culture side panel and some internal provenance metadata). This modelling is different from the si2 codebase which does not have the notion of `CultureGroups`. The Histri tree is stored in the database using the `parent` field on the si2 codebase `Culture`[9] entity. `CultureGroups` are stored using the same field, with the addition of the `parentIsSynonym` flag. Every synonymous strain number (`Culture`) from a `CultureGroup` has the first synonym (`Culture`) as its `parent` and the `parentIsSynonym` flag set to `true`. All annotations on CultureEdge are stored as fields on the child `Cultures`.

5.2.2 Selection and highlighting submodels

The `SelectionModel`[10] is a submodel that holds the current selection. The selection is changed through this model, and upon change, registered listeners (e.g. the

[8] `net.straininfo.histri.model.CultureEdge`
[9] `net.straininfo2.entities.legacy.Culture`
[10] `net.straininfo.histri.model.SelectionModel`

HistriPanel, the side panels and culture list) can update their view. Views that do not depend on selection do not register with this model. For example, the Histri tree layout does not need to be recalculated if the selection changes (however, the HistriPanel does need to be redrawn).

HighlightModel[11] enables to highlight cultures. Nine different highlight types (defined in HighlightType[12]) are supported, and for each type, each CultureGroup can be individually marked as highlighted or not. Depending on the type of highlighting, highlighted CultureGroups are drawn with a different colour on the HistriPanel. The small change in colour when hovering over a block is also implemented using this submodel (HighlightType.HOOVER). Although HighlightTypes other than HOOVER are not often used in practice, this feature is more important than it might seem at first sight. During the development of the suggestion system, it was used to visualise the cultures with a suggestion. Seeing that an increasing number of cultures had a significant suggestion, this eventually lead to the development of the autobuild algorithm.

5.2.3 Suggestion system and autobuild

The SuggestionManager[13] is the submodel that is responsible for calculating suggestions, keeping track of provenance and performing autobuild. At the core of the suggestion system is a data structure with base class Suggestion[14]. Formally speaking, a Suggestion could be defined as:

Definition 1. *A suggestion* s *of a relation between child* c *and parent* p *is a 6-tuple* $(c, p, \varepsilon, \delta, \alpha, \rho)$ *where* $\varepsilon \in \{\leftarrow, =\}$ *denotes the type of relationship,* $\delta \in \mathbb{R}$ *assigns a confidence value,* α *is a bag of possible edge annotations and* ρ *represents the provenance of the suggestion.*

In this definition, the following symbols are used:

c, p The atomic cultures (Cultures) of the child and parent involved.

ε The symbol $\leftarrow$ represents a normal edge, denoting a physical transfer (represented in the HistriModel as a CultureEdge[15]). The symbol $=$ represents a

[11]net.straininfo.histri.model.HighlightModel
[12]net.straininfo.histri.model.HighlightType
[13]net.straininfo.histri.model.suggest.SuggestionManager
[14]net.straininfo.histri.model.suggest.Suggestion
[15]net.straininfo.histri.model.suggest.CultureEdge

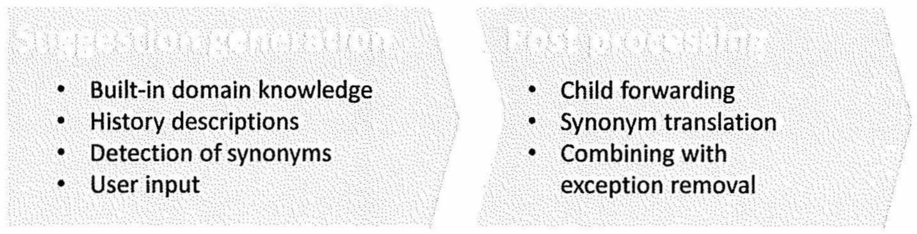


Figure 5.3: Generation and post-processing of `Suggestions` in `SuggestionManager`.

'parent is synonym' edge, meaning that both cultures belong to the same `CultureGroup`. Note that both types are directed (the order of c and p is important).

δ Assigns a confidence value to the suggestion. Convention is to take $\delta = 1$ for each 'atomic' origin.

α A data structure containing possible annotations for the suggested `CultureEdge`.

ρ Provenance: information about the origin of the assertion.

`SuggestionManager` generates `Suggestions`, which are used for the red border visualization of `HistriPanel` and by the autobuild algorithm. This is done in two phases. First, there is a generation phase, which generates suggestions based on a number of heuristics. All generators implement the `SuggestionGenerator`[16] interface. Second, there is the post-processing phase that bundles the generated `Suggestions`, by aggregating confidence values. An overview is shown in Figure 5.3. The generation phase consists of the following suggestion generators:

Domain knowledge. Based on historical domain knowledge (see Figure 4.2), this heuristic determines whether known acronym changes or other synonyms occur in the strain. For each strain number that is known to be the result of a known institutional transfer, it will try to find the predecessor as suggested by the domain knowledge.

Synonym without URL. This heuristic generates suggestions for non-unique strain numbers which it considers as synonymous. As the Histri editor was written

[16]`net.straininfo.histri.model.suggest.SuggestionGenerator`

before development on the si2 codebase was started, the notion of 'unique cultures' was not yet easily accessible. Therefore, the fact that a culture has a BRC catalog entry URL is used as a heuristic to approximate this. Non-unique strain numbers are split into tokens and if one of the tokens (which has a required minimal length and does not occur on a blacklist of invalid tokens) matches another strain number, the strain numbers are considered synonymous and a 'parent is synonym' suggestion is generated. This heuristic also generates suggestions for strain numbers that are highly similar (equal after limited normalisation or after removing a type strain 'T' postfix).

Textual history descriptions. This suggestion generator uses the textual history descriptions (given that they are stored in the StrainInfo index), and tries to match strain numbers to the parts in the textual descriptions. Based on those matched strain numbers, `Suggestions` are created. As the nature of these textual descriptions is very diverse, there are many heuristics necessary to be able to reach an acceptable level of precision and recall. Therefore, this is the most advanced heuristic suggestion generator used by the Histri editor. The complete system was conceived as the result of manually reconstructing Histris and looking at the data, resulting in incrementally discovering that there were widespread patterns that still could be automated.

`StrutSuggestionGenerator`[17] is a wrapper class that generates `Suggestions` based on `StrutManager`[18], the central class responsible for analysing all history descriptions, matching the strain numbers and other metadata to all parts of the history description string. In general, the history description string consists of multiple fields, separated by delimiters such as `<--`, `<` or `;`. We call these fields `Struts`[19], a name coincidentally similar but not to be confused with the web framework[20] (see Section 3.2.2). `StrutManager` splits the history description string into `Struts` based on a heuristically determined delimiter. Consequently, the resulting `Struts` are in a heuristical, incrementally fuzzy way matched with the set of strain numbers in the strain until one or more matches are found. First, the literal strut is matched against all strain numbers. If there are no matches, this means that there might be additional

[17]`net.straininfo.histri.model.suggest.StrutSuggestionGenerator`

[18]`net.straininfo.histri.model.suggest.StrutManager`

[19]`net.straininfo.histri.model.suggest.Strut`

[20]As already mentioned, the development of the si2 codebase was started after development of the Histri editor. The name was chosen to indicate that it concerns a small field of the history description string that has additional metadata.

information that is contained in the strut which needs to be identified. This includes year numbers or more precise dates at the beginning or end of the strut, and a depositor name. If, after cleaning the identified meta-information, the resulting strut still does not match strain numbers, the strut is split into words, which (if they meet some conditions) are matched against the set of strain numbers. It is possible that multiple strain numbers match (for example if the strut only contains an acronym). All matches and discovered meta-information are stored in the `Strut` object. For each `Culture` that has a textual history description, a `List` of `Struts` is held by `StrutsManager`. The 'history provenance' tab of the Histri editor displays the mappings found by the `Struts`.

`StrutSuggestionGenerator` generates the `Suggestion` objects based on all consecutive pairs of `Strut` objects. As all `Strut` objects can have multiple matching strain numbers, a `Suggestion` object is generated for all combinations of matching strain numbers (i.e. the Cartesian product of the consecutive `Strut`'s matching strain numbers). In addition, the 'atomic' confidence attributed to each pair of `Struts` is evenly distributed over all pairs of matching strain numbers. As the first `Strut` often contains the immediate predecessor of the culture the textual history description belongs to, `Suggestions` for this initial pair are also generated.

Edge caching. Edges already stored in the database are cached for autobuild. This is important as autobuild first disassembles the entire Histri tree before rebuilding. If there is no other provenance of that edge, the edge would otherwise be lost. These edges are stored as imported edge `Suggestions` with a high confidence value.

Dangling edges are edges of which the parent is not (yet) linked in the database, but already contains annotations, including the strain number of the parent. This is an artifact of the legacy screen scraping of some BRC catalogs. Dangling edges are converted to `Suggestions` if their strain number matches one of the strain numbers of the strain, so that they are reconsidered by autobuild (to link the parent).

After generating `Suggestions`, there are some post-processing steps to bundle suggestions:

Child forwarding. Implemented in `ChildForwarderPostProcessor`[21], this post

[21]`net.straininfo.histri.model.suggest.ChildForwarderPostProcessor`

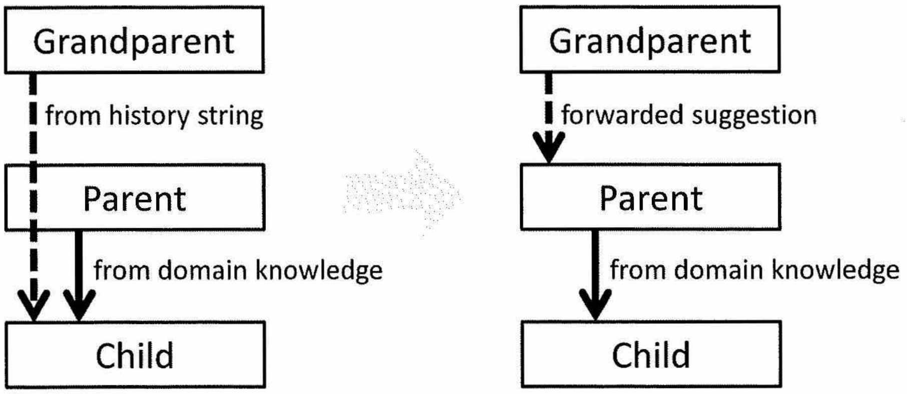


Figure 5.4: Child forwarding: heuristic that positions a domain knowledge edge 'between' suggestions found from history strings. Grandparent is actually the predecessor of parent, while the history description string suggests it is the predecessor of child.

processor is designed to handle suggestions from textual history descriptions that were not updated correctly after an institutional transfer. If a BRC changes the form of their strain numbers, but does not insert the old strain number in their history descriptions, `StrutSuggestionGenerator` will generate a suggestion from the historic predecessor to the new strain number. However, as can be seen on Figure 5.4, this suggestion actually 'skips' the original strain number. This is repaired by child forwarding: the `Suggestion` from the history string is replaced by a new `ForwardSuggestion` (encapsulating the original `Suggestion`) from the predecessor to the old strain number.

Interestingly, as some BRCs already went through two strain number changes, this post processor is run twice. In this case, there is an additional *middle parent* culture which also has a domain knowledge suggestion to parent. The second iteration forwards the forward suggestion between parent and grandparent to middle parent and grandparent. It is not necessary to run this post processor more than twice as there are no instances known of three or more consecutive institutional changes (see Figure 4.2).

Synonym translation. The fact that `CultureGroups` can group multiple atomic `Cultures` and that `CultureEdges` are inserted between `CultureGroups` complicates the aggregation of confidence and autobuild. Therefore, all Suggestions applying to Cultures that are not the first Culture of their CultureGroup, are translated to the first Culture of the CultureGroup they belong to. This is implemented in TranslatePostProcessor[22].

Combining. This post processor, implemented in CombinePostProcessor[23], combines different suggestions between the same Cultures into one Suggestion. In this process, an AggregationFunction[24] is used to combine the confidence values of all underlying Suggestions, which are saved as provenance in the resulting CombinedSuggestion. The default AggregationFunction used is a simple sum, but an implementation of the MYCIN and Prospector aggregation functions are also added (note that the confidence values generated by these functions are in the interval $[0,1]$, as opposed to sum, which operates in $\mathbb{R}$). The same AggregationFunction is also used by StrutSuggestionGenerator to divide the 'atomic' confidence.

CombiningPostProcessor is also able to mask conflicting suggestions. Some suggestion generators generate a small fraction of incorrect suggestions, mostly as the result of partial information, with the idea that these inconsistent suggestions can be filtered out later without sacrificing recall when having a bird's eye view. Conflicting Suggestions occur when there are Suggestions which suggest different predecessors for a given CultureGroup, i.e. when having suggestions $(c, p_i, \varepsilon_i, \delta_i, \alpha_i, \rho_i)$ with $1 < i \leq n$ and $(\exists i,j)(p_i \neq p_j)$. In this case, CombiningPostProcessor compares the confidence of those Suggestions and determines whether there is a single suggestion that has a majority of the confidence compared to the aggregated confidence of all other Suggestions. If true, the inconsistent Suggestions are added as negative provenance to the first Suggestion and the confidence is subtracted from the total confidence.

After each model change, the suggestions need to be recalculated as a model change can have a large impact on the output of some generators and post processors. This calculation is only performed on demand so that two consecutive model changes not necessary result in two suggestion recalculations. The generated Suggestions are used by HistriPanel (suggested predecessors for the selected

[22]net.straininfo.histri.model.suggest.TranslatePostProcessor
[23]net.straininfo.histri.model.suggest.CombinePostProcessor
[24]net.straininfo.histri.model.suggest.aggregators.AggregationFunction

`CultureGroup` are drawn with a red border) and by autobuild (as input for the algorithm). From the perspective of adding autobuild functionality, much work is already performed by the suggestion generation pipeline, and therefore the effective autobuild algorithm itself is relatively easy. The autobuild algorithm takes a set of `Cultures` that need to be built and works as follows:

1. If building the complete Histri, the current tree is disassembled, to be able to build the complete tree from scratch.
2. All 'parent is synonym edge' suggestions are processed: all synonymous atomic `Cultures` are merged into `CultureGroups` (only if at least one of the `Cultures` of the processed `Suggestion` belongs to the set of `Cultures` that are built).
3. The suggestion cache is rebuilt as the changed `CultureGroups` can have a large impact on the results of `TranslatePostProcessor` and `Combining-PostProcessor`.
4. For each `Culture` in the build set, a list of possible `CultureEdges` (with their suggested annotations) is created. The order in which this is done is not important as each `Culture` has at most one parent.
    (a) If only one possible `CultureEdge` is found, the edge is inserted, and the annotations are added to the edge.
    (b) If multiple possible `CultureEdge` are found, no transfer is inserted and information about the conflicting possibilities is saved.

The build set is used to control the 'application area' of the algorithm. The flexibility in the application area also allows to build a small subset of `Cultures` or even a single `Culture`. For example, in the Histris shown on the strain passport, only the `Cultures` with no relation to the tree are built to make sure that the edges stored in the database are given priority over adding additional or different edges which may be suggested. When using autobuild in the Histri editor, the complete reconsideration of all evidence is desired. The Histri editor also has a feature to only build the selected `CultureGroup`.

5.2.4 Undo functionality

Good editors include undo functionality and users have become accustomed to it. It motivates users to experiment as it reassures them that they can recover from unwanted changes. Undo functionality is implemented in UndoManager[25]. Built on the fine-grained event dispatching, UndoManager is actually relatively easy to implement for the majority of operations on the HistriModel.

UndoManager holds a stack of all modification events from HistriModel containing a type (the kind of operation that has been performed) and a reference to the object that has been changed. Undoing an operation is implemented as doing the inverse of the event at the top of the stack. The information attached to the event is used to determine which operation needs to be undone, and the object on which it applies. For example, if a transfer is inserted, the event will contain a reference to the CultureEdge that was inserted. To undo this operation, the CultureEdge needs to be removed from the HistriModel. Subsequently, the undone event is removed from the undo stack and moved to the redo stack, to be able to redo if needed.

In terms of the data model, only operations that have an inverse can be easily undone: adding vs. removing a culture, inserting vs. removing an edge, make invalid vs. make valid and new meta-information vs. old meta-information. Other operations have a larger impact on the model, and need more data to be able to restore the data model into its initial form and therefore do not have an undo implementation. An example of this is adding a strain number as a synonym of another (i.e. merging two CultureGroups). If an event is fired that can not be undone, both the undo and redo stack are cleared. UndoManager only listens to events from HistriModel, and ignores all events from other submodels as they do not modify the Histri tree.

The code that performs the inverse operations is located in the UndoManager class. The downside of this is that the addition of new operations to the HistriModel incurs changes to UndoManager to add undo support. Undo functionality was added late in the development process, but an architecturally better alternative would have been to use the Command design pattern. Herein, the code that performs the actual modifications to the data model is then added to the events, which also contain state information to be able to do and undo their intended operation. The events are then referred to as commands and this reduces the implementation of UndoManager to merely keeping track of the command objects.

[25] net.straininfo.histri.model.UndoManager

5.3 Histri panel design

The central GUI component of the Histri editor is the Histri panel. The panel displays (and layouts) the current Histri, but at the same time enables editing by allowing to modify the tree by dragging and dropping blocks. All classes belonging to the component are grouped in package `net.straininfo.histri.gui.histripanel`. The class bringing everything together is `HistriPanel`[26], a subclass of `JComponent`[27]. Although we have experimented with diagramming and graph frameworks, the particular functionality required resulted in better designed and more efficient code when written from scratch. In addition, due to the complexity of the component, some aspects were pulled apart and developed independently of each other. This explains why certain aspects are abstracted to separate subsystems.

5.3.1 States

The Histri panel supports highlighting, selecting and deselecting of blocks, dragging blocks and dropping on others to insert links and scrolling when dragging the background. After starting an *ad hoc* implementation of the `MouseInputListener` and `MouseWheelListener` interfaces, it became apparent that a proper model was necessary in order to be able to implement this functionality properly and without (hidden) bugs. As a consequence, a model consisting of seven different states was conceived. Each state corresponds to a particular configuration of the panel, depending on whether there is a selection, whether the pointer is hovering above a block or the selected block, and whether the mouse button is pressed down (clicking and dragging). An overview of all states and what they mean intuitively is given in Table 5.1. Naturally, states only make sense if it is possible to transition between them, and an overview of the different transitions between states are shown on Figure 5.5.

Many states have transitions where the end result depends on the position of the mouse pointer, i.e. whether it is hovering over a block, the selected block or over white space. This decision is independent of the preceding state. To simplify the transition scheme, two virtual nodes that make an immediate decision based on the position of the mouse pointer were introduced. They make the transition scheme less complex, and also make code more compact by allowing reuse of the decision code. The decision depends on whether or not there is a selection, and

[26] `net.straininfo.histri.gui.histripanel.HistriPanel`
[27] `javax.swing.JComponent`

therefore there are two nodes. These nodes are also the starting point when the mouse enters the component for the first time.

5.3.2 Tree layout

Before the Histri tree can be drawn, it is necessary to calculate the layout, i.e. to assign each block (`CultureGroup`) a position on the screen so that the resulting tree is correctly laid out, compact and does not have undesirable overlaps. This is not a trivial task and given the many possible algorithms and approaches, the tree layout subsystem was architecturally separated from `HistriPanel`, so that new layout algorithms can be added when required.

`BlockLayout`[28] is responsible for keeping track of the position of all `CultureGroups` on the `HistriPanel`. It stores the current location of each `CultureGroup`, obtained through `getLocation(CultureGroup)`. Upon calling the `layout()` method, the tree layout is calculated (i.e. all `CultureGroups` are given a position) and cached. The actual layout algorithm is an implementation of the `BlockLayoutAlgorithm`[29] interface. The current release of the Histri editor uses `LayerFillLayout`[30]. This layout uses a discrete grid approach (with fixed size columns but variable row heights) and moves subtrees as close as possible to (or in between) each other on the grid. Using `getPreferredLocation(CultureGroup)`, the calculated layout can be retrieved. It is the responsability of `HistriPanel` to change the current location of the `CultureGroups` to the preferred location calculated by `BlockLayout`. The advantage of this is that `CultureGroups` do not immediately 'jump' to their preferred location and that the relayout process can be animated.

5.3.3 Animation

The Histri editor uses the Timing Framework[31] (classic distribution, version 1.1) of Chet Haase [70] to animate the `HistriPanel`. This framework is centered around `Animator`[32] objects, that contain properties of the animations they represent, and are responsible for 'driving' the animation. Therefore, animations can be started and stopped by calling corresponding methods on this class. While the animation is running, the class fires animation events. These events are caught

[28]`net.straininfo.histri.gui.histripanel.layout.BlockLayout`
[29]`net.straininfo.histri.gui.histripanel.layout.BlockLayoutAlgorithm`
[30]`net.straininfo.histri.gui.histripanel.layout.LayerFillLayout`
[31]`http://java.net/projects/timingframework/`
[32]`org.jdesktop.animation.timing.Animator`

State name	Selection	Hoovering over block	Hoovering over selection	Dragging	Comment
PASSIVE	no	no	no	no	no selection and/or highlighting
PASSIVE_WITH_SELECTION	yes	no	no	no	selection, no highlighting
HIGHLIGHT	no	yes	no	no	hovering over block, no selection
HIGHLIGHT_SELECTION	yes	yes	yes	no	hovering over selected block
HIGHLIGHT_WITH_SELECTION	yes	yes	no	no	hovering over block other than selection
DRAG_SELECTION_L	yes	yes	yes	yes	dragging started on selected block (left button)
DRAG_SELECTION_R	yes	yes	yes	yes	dragging started on selected block (right button)

Table 5.1: Overview of the different states of the Histri panel.

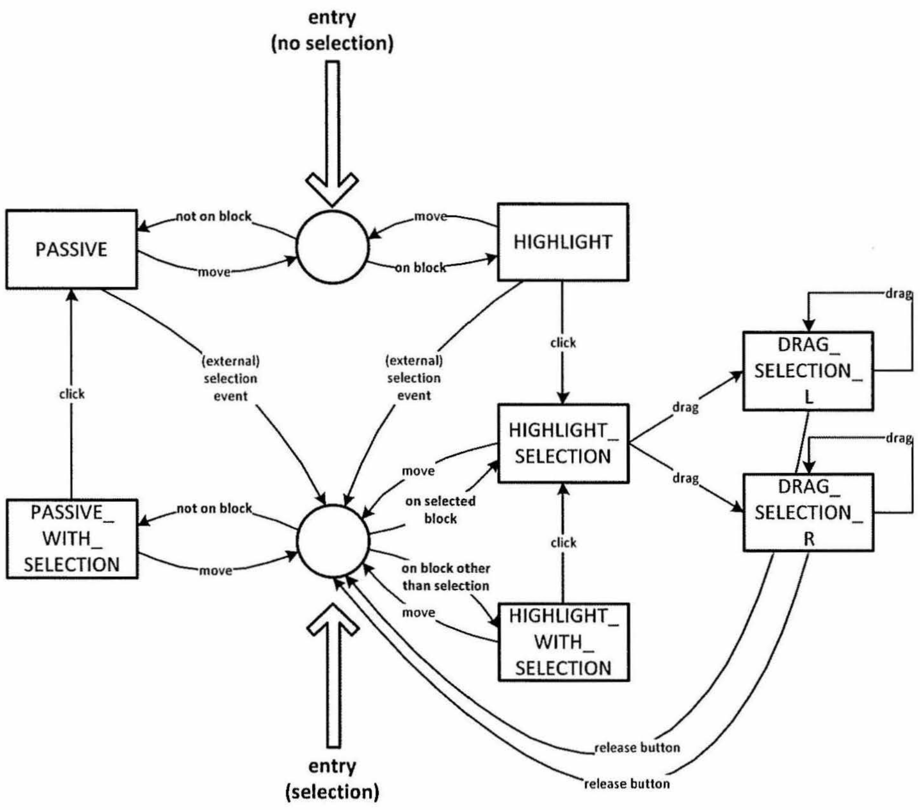


Figure 5.5: Transitions between states of the Histri panel.

by TimingTarget[33] implementations, which form the connection between the animation running on Animator and the animation actually doing something. The listener methods in TimingTarget are given information about the current animation state and can set up object state, calculate new property values, or do anything else appropriate for the situation based on for example an interpolation using the animation state. HistriPanel has three animated aspects:

Animation of tree layout changes. This occurs when the tree is modified and the layout changes. After a tree change, the tree layout algorithm calculates new coordinates (BlockLayout.getPreferredLocations()) after an explicit call from HistriPanel. The animation is then started by creating a new RelayoutTimingTarget. For the duration of the animation, intermediate locations are calculated between the current coordinates (copied into startLocations) and the preferred layout (preferredLocations).

Animation of automatic scrolling. This occurs when a culture is selected that is currently not visible because of scrolling. Animation is used to smoothly move the viewport on the HistriPanel so that the selected culture becomes visible.

Animation of zoom. This occurs when a zoom level is changed. For the duration of the animation the zoom level is gradually interpolated.

5.3.4 Themeable Histri tree decoration

To enable theming of the Histri tree, most of the actual drawing code is abstracted from HistriPanel. Essentially, four different types of objects are drawn: CultureGroups (drawn as blocks), CultureEdges (drawn as lines), the separate background for CultureGroups that are not attached to the three and the small copyright tag at the bottom of the figure. Three of them are drawn using a plug-in approach: the block decorator (drawing the blocks), the edge decorator (drawing the line between blocks) and the tag decorator (drawing the copyright tag). By substituting them by another implementation, HistriPanel is able to change the theme at any time.

HistriPanel recursively draws CultureGroups, first drawing the top of the tree and if necessary the edge to its parent, recursively followed by all subtrees.

[33] org.jdesktop.animation.timing.TimingTarget

CultureGroups are rendered to a block by implementations of the BlockDecorator[34] interface. The actual painting is performed by the implementation of the paint method (paint(HistriModel, CultureGroup, Graphics2D)) on that interface. As the size of the rendered blocks is of importance for the tree layout algorithms, implementations also need to return the size of a rendered block corresponding to a CultureGroup in the getBounds(CultureGroup) method. Changing the BlockDecorator requires the recalculation of the tree layout, as the rendered size of CultureGroups can change.

The lines between blocks are drawn by implementations of EdgeDecorator[35]. This interface similarly has a paint method (paint(HistriPanel, CultureGroup, Graphics2D)) which draws the line from a CultureGroup to its parent. The default implementation, OrthogonalEdgeDecorator, draws the currently used typical horizontal lines with varying width.

TagDecorator[36] implementations draw the small copyright tag seen in PDF and PNG versions of the Histri tree. This is not done when using the HistriPanel interactively in the Histri editor, which uses a void tag decorator (NoTagDecorator), as opposed to the standard BasicTagDecorator.

5.4 Authentication transfer

In order to protect StrainInfo from vandalism and to be able to credit people for their work, the Histri editor can only be used after logging in. Users log in using the log-in mechanism provided on the StrainInfo web application. As the Histri editor is launched as a stand-alone program, it is necessary to transfer the authentication of the already authenticated user to the Histri editor. This is preferred over asking users to reauthenticate when the editor loads the current Histri or at the time of saving, as this might decrease the user's willingness to actually curate and save Histris.

The StrainInfo web application uses a jsessionid cookie to allow sessions. This is the default cookie used by JSP to enable sessions. At server side, the log-in mechanism adds the authenticated user to the session object (which holds data the web application might have stored about the session of the user). After this cookie has been set by the web server, the client (i.e. the web browser) will set this

[34]net.straininfo.histri.gui.histripanel.blockdecorator.BlockDecorator
[35]net.straininfo.histri.gui.histripanel.edgedecorator.EdgeDecorator
[36]net.straininfo.histri.gui.histripanel.tagdecorator.TagDecorator

cookie as part of the HTTP header in subsequent requests. This way, server-side code is able to identify the user in the request, and show pages tailored to that specific user. The Histri editor JNLP file and the Histri web services are protected by authentication. When downloading the JNLP file, the web browser is responsible for setting the cookie. However, in order to use the Histri web services, the Histri editor also needs to be able to set the cookie in order to prove that the web service request comes from the same user. Therefore, it needs to acquire the value of the `jsessionid`. The authentication is transferred to the Histri editor by transfering the `jsessionid` cookie value to the editor. This is done in the JNLP file, using the property `histri.jsessionid`. This value is read by the Histri editor which sets the cookie by all calls to the web services (which are HTTP requests).

The JNLP file also contains the property `histri.username` which contains the user name corresponding to the cookie. In theory, this is not necessary, as when saving the Histri, the server knows the user associated with the cookie and therefore can link the saved version to a user. However, due to historical reasons[37], the Histri editor also saves the username in the XML file it sends to the server upon saving. In addition, this allows to analyse the saved XML files without additional database access to collect user metadata.

A side effect of this authentication mechanism is that the user needs to stay logged in in the web browser. If the user logs out, the `jsessionid` becomes invalid, and therefore loading or saving a Histri will fail. Therefore, the Histri editor contains a log in feature.

It is clear that using cookies for user authentication and sessions poses risks for man-in-the-middle attacks. It is possible to use more secure protocols (in fact the authentication request itself is performed using HTTPS), but for various reasons cookies are very often used in practice for authentication. In order to make it more difficult to take over the session of another user, the value of the `jsessionid` cookie is changed after successful authentication, to prevent so-called 'session-fixation' attacks.

[37]The initial version of the Histri project homepage, implemented before the si2 codebase was conceived, used an authentication mechanism provided by Ghent University ('webauth'). This was used to test Make Histri by lab members, before it was made available to the public.

Part III

Adding semantic information to StrainInfo

6 The Microbiological Common Language

The Microbial Commons designates various initiatives that attempt to move towards a globally distributed research infrastructure, based on facilitated access to biological material, related data and information for basic research, education and commercial use purposes [71, 72]. Part of the Microbial Commons is seeking a way of electronically exchanging information about microorganisms, leveraging a lingua franca for disseminating microbial information and meta-information in online environments. Although there have been several standardization and integration attempts in the past, these have failed to set up a standardized, open infrastructure that allows electronic processing (see Section 2.2.2). Although relatively new, StrainInfo has quickly become the *de facto* reference for information and meta-information on microorganisms. However, this information still needs to be made available in a common electronic format. Therefore, next to the legal and governance aspects of a Microbial Commons, a common language for the exchange of microbial information is essential to lift microbial research to the next level of comparative *in silico* analysis.

6.1 Microbiological Common Language (MCL)

The Microbiological Common Language (MCL) has been designed to facilitate the electronic exchange of information on microorganisms. It allows capturing information available in BRC catalogs in an electronic form. The standard is viewed as a framework for the rich description of microbial material ranging from information on the sampling and isolation process and availability in BRCs to the biochemical properties of the microbial material being described. However, there is no clear boundary on what is reasonably seen as being 'the fields required to fully describe microbial material'. Therefore, the standard consists of an abstract model which establishes a generic framework, together with a basic set of terms that facilitate capturing everyday BRC catalog contents. This allows translating BRC catalogs into this new format using transparent mappings from existing fields. Moreover, the framework also allows the use of specialized terms which come, as a consequence of their specificity, from other standards.

The standard has been heavily influenced by the Microbial Information Network in Europe (MINE) vocabulary [40, 41], and implements the recommendations put forward by the OECD [17]. The MINE standards were early attempts to standardize BRC databases and rigorously define data fields and their application range. However, as MINE also defines the underlying database structure, this imposes a heavy burden on implementing the standard within a BRC. Therefore, MCL is conceived as a communication standard rather than an internal database schema. Moreover, the abstract model of the standard is loosely coupled with actual implementations. This means that the standard meticulously defines the name, definition and structure of its elements, but not how to syntactically format the data. The advantage of this approach is that the standard does not depend on a particular representation technology and that it can be easily ported to new data transfer systems. For practical usage, however, it is necessary to agree upon a particular implementation, and therefore concrete usage is discussed in Section 6.3.

Along with the loose coupling of abstract model and implementation, MCL is also novel in its approach to structuring microbial information. MCL uses the natural workflow from sampling and isolation to a description of the original isolate and subsequent deposits in BRCs (see Section 1.1). Traditional standards do not model these steps and use a flat list of properties and their values instead. The model is visualized in Figure 6.1 and contains seven cornerstone 'entities' that represent classes of objects that can be described using the standard and resembles a somewhat simplified version of Figure 1.2. By convention, entity names start with

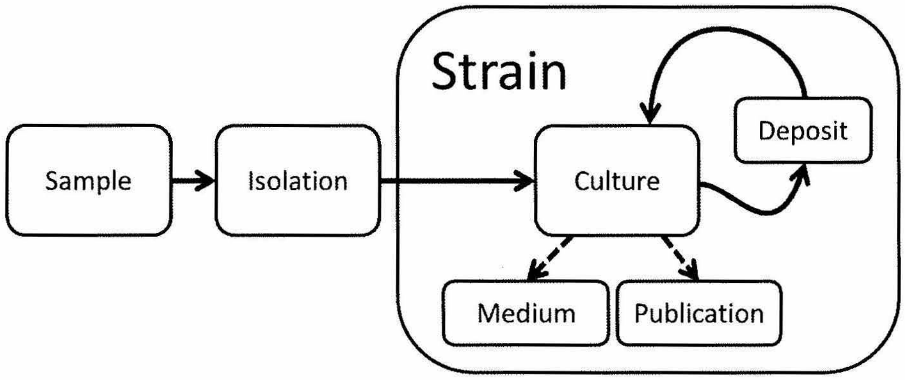


Figure 6.1: The abstract model of MCL follows the logical flow from sampling to subsequent deposits in BRCs.

capital letters. The logical microbiological workflow starts with a physical `Sample` taken from the environment, followed by an `Isolation` process which results in a `Culture`. Cultures can then be subsequently `Deposited` in BRCs to gain worldwide distribution. Moreover, cultures have associated `Publications` and BRCs often list recommended growth `Mediums`. These processes and results are independently described by the standard.

For each entity, several 'terms' are defined that can be used to describe its properties. Entities correspond to 'things', whereas terms correspond to 'properties' of those things. By convention, the names of terms start with a lowercase letter. A few examples of the MCL terms can be found in Table 6.1. Although most terms have a self-explanatory meaning, all terms are precisely defined by the language reference which is available online[1] (see also Appendix A). The online document contains the exact definition for all terms and is the authoritative term definition and description of term usage.

6.2 Relation and compatibility to other standards

The MINE project performed pioneering work in structuring information on microorganisms. Originated in the late eighties, the original fungal [41], and later

[1]`http://www.straininfo.net/projects/mcl/reference/`

Term – Definition [MINE]	Entity
`strainNumber` – A strain number used to identify a culture or strain. [STN, ACCN]	`Culture, Strain`
`otherStrainNumber` – Equivalent strain number. Not to be used as an identifier for cultures. [OCC]	`Culture`
`speciesName` – Species name of the organism. [SP]	`Culture, Strain`
`typeStrainOf` – Name of taxon for which strain is type strain [AT]	`Culture, Strain`
`sampleLocationCountry` – Country where sample was taken. [LOC]	`Sample`
`sampleHabitat` – Textual description of habitat where sample was taken. [ISOFR]	`Sample`
`isolationDate` – Date of isolation [ISOL]	`Isolation`
`isolator` – Isolator; person responsible for isolation from sample [ISOL]	`Isolation`

Table 6.1: For each term, a short definition and an indication (in square brackets) of a corresponding MINE field is given. The last column indicates the corresponding entities. For the full and authoritative definition of the terms, we refer to the language reference.

bacterial [40] MINE standards defined a database schema along with an extensive list of fields conceivable at that time. Many of the MCL terms are based on MINE fields, and if applicable, the related field(s) are indicated in the language reference document. MINE defines a Minimum Data Set (MDS) of fields to be used for the exchange of information between the national nodes of the MINE network and in printed catalogs. As an extension of the initial goals of the MINE framework, CABRI was the first initiative for integrating multiple BRC catalogs into a single virtual catalog. Next to the standardization of information validation and dissemination, and the linking between catalogs, it also aims to guarantee BRC quality by the creation and enforcement of quality management guidelines and standards. The database includes 28 catalogs covering bacteria, archaea, fungi, yeasts, plasmids, phages, DNA probes, animal and human cell lines and plant cells and viruses. The search engine, which incorporates queries on species name and strain numbers, is able to search through all catalogs at once or through individual catalogs. The system includes a uniform front-end for making the catalogs of multiple BRCs accessible as a one-stop shop. The database model of CABRI was heavily influenced by MINE and many of the CABRI terms exactly follow the MINE format and syntax. CABRI also adopted the notion of an MDS, but additionally defines a Recommend Data Set (RDS) and a Full Data Set (FDS).

MINE (and thus also CABRI) fields often contain multiple pieces of information, which are frequently demarcated using a peculiar syntax. In consequence, current BRC catalogs still contain fields that combine several distinct pieces of information (MINE uses the term 'subfields') into a single field. This poses problems for automatic processing, as these fields ideally need to be split into their atomic components. To overcome this problem, terms that are specifically designed to match these legacy fields were added to MCL. These terms are most often marked as deprecated, but were nevertheless added to allow BRCs to export semantically rich files without the initial need for manual data curation. An example of this type of curation is splitting a free text sample description into a separate habitat, sampling country and place field, a task which is difficult to carry out algorithmically. Moreover, the language uses terms from the PRISM (Publishing Requirements for Industry Standard Metadata) standard[2] for the fine-grained description of publications. Dublin Core[3] is used to include a 'human understandable' (and thus not normalized) citation (see Guidelines in Dubin Core[4]). However, this by no means

[2]http://www.prismstandard.org/

[3]The Dublin Core Metadata Element set, see http://dublincore.org/documents/dces/

[4]http://dublincore.org/documents/dc-citation-guidelines/

signifies that normalization is not necessary. Data normalization is one of the goals of MCL. These fields enable postponing the normalization work to a later stage, which in practice means that StrainInfo will have to develop means to normalize the legacy data and thus improve the recoverability and retrievability of data from BRCs.

6.2.1 Compatibility to Genomic Standards

Genomic and post-genomic analysis of microbial strains has become more routine and widespread in recent years. The genomic community became aware that a standardized rich set of meta-information accompanying DNA sequence data is of importance to better facilitate comparative genomics studies. The Minimum Information about a Genome Sequence (MIGS) checklist published in 2008 [73] captures the complete context of a genome sequence and defines the minimum information needed to better facilitate comparative studies. This information includes geographic and spatial details of the original sample, information on cultured organisms and details of the experimental techniques used. MIGS is also implemented in the Genomic Contextual Data Markup Language (GCDML) [74] using XML.

Especially in the description of cultured organisms, there is quite some overlap between MIGS and MCL. To align and establish a strong bond between both standards, MCL adopts the same modular approach as already used by GCDML (see Figure 6.1). In turn, this allows GCDML to incorporate the MCL terms which overlap MIGS. This transparent alignment of MCL and GCDML is technically possible due to the capacities of XML to consistently mix vocabularies. This would not be possible with the formats defined by MINE and CABRI.

6.3 Usage and Implementations

Although the previous section describes the overall structure of the proposed standard, we have not yet described its practical application. As the structure of the standard intuitively maps to the XML Document Object Model (DOM), XML has been chosen as the default representation technology. It is equally possible to represent the data using RDF: the RDF/XML representation is highly similar to the plain XML representation. Each distinct use case of the standard has a corresponding 'schema'. Schemata enforce strict formatting, term order and also define obligatory fields. Documents can be automatically validated against the schema, which

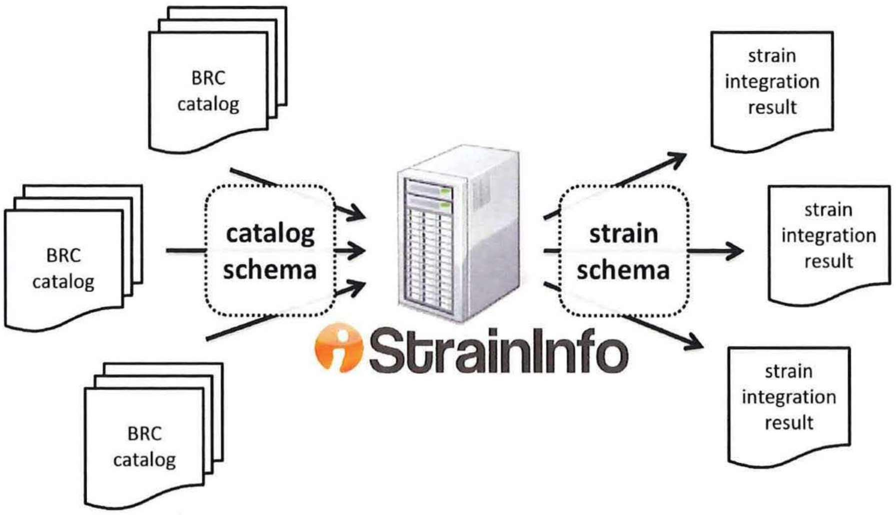

BRC catalog
BRC catalog
BRC catalog
catalog schema
StrainInfo
strain schema
strain integration result
strain integration result
strain integration result

Figure 6.2: Two distinct use cases of MCL. At the left-hand side, MCL is used to synchronize BRC catalogs with StrainInfo (using the catalog schema). This information is integrated by StrainInfo, and the integration results are made available in MCL documents formatted according to the strain schema (right-hand side).

facilitates error detection and reliable electronic processing. Therefore, a distinct schema is developed for each particular use case, in order to meet the demands of the application at hand. A full reference of some implementations, including links to the corresponding XML and RDF/XML schemata, can be found at the MCL project home page[5].

An important use case is the representation of BRC catalogs. This is implemented by the 'catalog' schema[6], which contains specific guidelines for exporting BRC catalogs. The schema defines the 'minimal set' of required (obligatory) terms and introduces a header which contains meta-information on the export itself. The exports are used by StrainInfo in its synchronization procedure as shown in Figure 6.2. Currently, sixteen BRCs have already adopted this format and synchronize with StrainInfo by regularly uploading an MCL export of their catalog (see Table 6.2). In this procedure, BRCs push their updates to StrainInfo instead of

[5]http://www.straininfo.net/projects/mcl
[6]Located at http://www.straininfo2.ugent.be/schema/2.0/si-catalog.xsd

Acronym	WDCM number	BRC Name	Country
CFBP	639	Collection Francaise des Bacteries Phytopathogenes	France
CIP	759	Collection de L'Institut Pasteur	France
CNCTC	130	Czech National Collection of Type Cultures	Czech Republic
LMG	296	BCCM^{TM}/LMG Bacteria Collection	Belgium
PCC	481	Pasteur Culture Collection of Cyanobacteria	France
VKM	342	All-Russian Collection of Microorganisms	Russia
VTT	139	VTT Culture Collection	Finland

Table 6.2: List of early adopters (alphabetical order). These BRCs synchronize with StrainInfo using MCL since October 2009.

StrainInfo pulling the catalog data at regular time intervals. This allows timely and accurate updates and enables BRCs to control their presence in the StrainInfo index. At the same time, this also solves the scalability problems of the pull paradigm. As BRCs indexed using the pull paradigm require the development and continuous maintenance of custom parsers required for screen-scraping their catalogs, this eliminates the need for time-consuming manual interventions when performing index updates. Moreover, by possibly offering the files available for download separately, BRCs can allow external resources to electronically access their catalogs independently of StrainInfo.

Another use case illustrated in Figure 6.2 is the export of the StrainInfo integration itself. StrainInfo receives information on cultures from multiple resources and performs integration at strain level. This integration process bundles all information known about a given strain, i.e. all cultures and all culture-related information. The integration process consists of linking equivalent cultures, the necessary error detection and correction and the semantic aggregation of information attached to the cultures. Although the resulting information is made available to the end-user through strain passport pages, well-formatted MCL documents are necessary to be able to further process these integration results automatically. The strain export (defined by the 'strain' schema) populates a full `Strain` element which includes the integration result as well as the original data (Cultures). A stripped-down example is shown in Figure 6.3. The example displays a strain listing three equivalent strain numbers, species and type strain information, and meta-information on the original biological sample and isolation. The example does not contain the original `Culture` elements. Note the usage of the `mcl` namespace to indicate that the terms are defined in the MCL context. The full declaration of the namespace is defined

```
<mcl:Strain xmlns:mcl="http://www.straininfo.net/ns/mcl/2.0/">
   <mcl:strainNumber>CCUG 27413</mcl:strainNumber>
   <mcl:strainNumber>CIP 103790</mcl:strainNumber>
   <mcl:strainNumber>DSM 4216</mcl:strainNumber>

   <mcl:speciesName>Bacillus smithii</mcl:speciesName>
   <mcl:typeStrainOf>Bacillus smithii</mcl:typeStrainOf>

   <mcl:Sample>
        <mcl:sampleLocationCountry>USA</mcl:sampleLocationCountry>
        <mcl:sampleHabitat>cheese</mcl:sampleHabitat>
   </mcl:Sample>

   <mcl:Isolation>
        <mcl:isolationDate>1947</mcl:isolationDate>
   </mcl:Isolation>
</mcl:Strain>
```

Figure 6.3: Example usage of the XML implementation of the standard. The example contains an export of a strain as integrated by StrainInfo. For clarity, detailed culture meta-information has been omitted. Terms are defined in the `mcl` namespace to indicate that the terms are defined in the MCL context. The full definition of the namespace is included as a `Strain` element.

in the `Strain` element. This document is intended to be consumed by downstream applications performing further analysis on the strain.

Furthermore, *ad hoc* use of the standard is accepted, but terms should be correctly formatted and employed. A typical example of this is in tables, where one or more MCL columns can be mixed with user-defined columns. By using MCL terms instead of free-form names as column headers, the column contents are strictly defined, and by consequence, this enables integration with other resources. For example, particular custom table exports in StrainInfo use certain MCL terms as column names.

6.4 Discussion

6.4.1 Applicability of the standard

MCL is intended to be broadly applicable in situations where microbial material is referenced or used and therefore has been designed to be interoperable with existing and future standards. Its use ranges from identifying particular cultures (with a minimal Culture element) to the full description of a strain (i.e. the integrated view of StrainInfo). Using the standard uniquely for referencing cultures (i.e. without appending additional meta-information) already allows interesting applications. For example, publishers could use the standard to list all cultures mentioned or used in their publications. This allows the accurate and complete indexing of publications and thus the reliable listing of all publications applicable to a certain strain or species. Analogously, GenBank/EMBL/DDBJ flat files could use MCL terms to link the sequence to the corresponding culture (strain number) from which it was taken. Adding more meta-information to the cultures increases the possibilities of downstream applications. For example, in order to be able to list all strains isolated from samples taken from a particular geographical region, sample descriptions that contain detailed geospatial meta-information are necessary. Moreover, this information needs to be available in a format suitable for computational processing, as it must be combined with a gazetteer to determine whether it is part of the geographical region. Building semantic search functions subsumes computational access to detailed meta-information along with the availability of supporting ontologies and reasoning technologies.

The standard is envisioned to form a framework for the description of rich, semantic microbial information. It is expected that more specialized vocabularies will make use of or naturally extend MCL to model more complex microbial information. Therefore, It is foreseen that the set needs to be augmented with new terms if proven necessary. However, this may be problematic, as existing terms cannot change their definition and the structure of the language has to remain unaltered. To allow future extension, the MCL namespace deliberately contains a version number. This version number reflects revisions of the standard and, if new terms are added to new versions, can also be used to distinguish between different implementation levels.

Note that MCL itself does not solve the issue of globally unique identifiers (GUIDs) [75]. This is solved independently of MCL by introducing the culture URI (see Section 2.2.4). However, culture URIs are highly compatible with MCL, especially in the RDF representation where culture URIs are used to explicitly refer to `mcl:Culture` resources. `Culture` elements in MCL XML files are anonymous and do not have an explicit identifier. Therefore, the term `mcl:cultureURI` can be used to include the culture URI in XML files.

6.4.2 Technical integration into StrainInfo

MCL is modelled in the si2 codebase, and this model is used to generate all derived products such as the MCL reference, RDF schema and XML exports of StrainInfo. The MCL model and all related functionality is bundled in package `net.straininfo2.ontology` in the core module of the si2 codebase. MCL entities and terms are stored as Java enum values in two enums (`Entity`[7] and `Property`[8]). The advantage of this implementation is that MCL can be used internally using type-safe symbols (instead of *ad hoc* `String` constants) which also give access to other aspects of the terms. For example, using the same enum value, the full URIs of the term can be used when generating XML or RDF, and a human readable form can be used when generating output for the end-user. In addition, only one authoritative source of MCL needs to be maintained in the si2 codebase, lowering the chance of inconsistencies and the general overhead of handling MCL. `Entity`s and `Property`s implement a common `Term`[9] interface (see Listing 6.1), which gives access to the term's fully qualified name and short-hands thereof. The si2 codebase uses the name 'term' internally as a collective name for both entities and terms (the latter internally being referred to as 'property'). In addition, each enum has additional functionality that is specific to the term's role.

[7]`net.straininfo2.ontology.Entity`
[8]`net.straininfo2.ontology.Property`
[9]`net.straininfo2.ontology.Term`

MCL entities are modelled in the `Entity` enum and roughly correspond (or map) to managed entities in the si2 codebase. The `Class`[10] of the managed entity is stored in the `Entity`, so that it can be used by mapping code in the si2 codebase. The entity corresponding to the ORM entity `Culture` is `Entity.CULTURE`, the enum representation of `mcl:Culture`. Other important entities include `Entity.STRAIN`, `Entity.SAMPLE`, `Entity.MEDIUM`, and `Entity.PUBLICATION`. The reference (see Appendix A) contains all defined entities and their definition.

MCL terms are modelled using the `Property` enum. All properties have an enum value: for example, the enum values `Property.STRAIN_NUMBER` and `Property.SPECIES_NAME` model the MCL terms `mcl:strainNumber` and `mcl:speciesName` respectively. This is shown in Listing 6.2, which contains an extract of the `Property` enum illustrating most situations. Because enum values have a one-to-one mapping to MCL terms, they can be used interchangeably. Properties have a domain and a range. The domain is the set of entities the property applies to. For example, the domain of `Property.SPECIES_NAME` is `Entity.CULTURE` and `Entity.STRAIN`, as this term is used in two different contexts (species name as listed for the culture vs. the integration result). The range of a `Property` applies to the type of its values. Some properties have literal values (e.g. a number, text, etc.), while others refer to other entities. The range of a property is the set of the types of the entities that the property can have as a value. In addition, `Property` also contains the definition of the 'sub property of' relation, whether it is functional and whether it has a `CultureTriple` representation (i.e. if the domain contains `Entity.CULTURE` and if the property is not stored as a field on a legacy entity). Following the definition given by the specification of the OWL Web Ontology Language[11], a functional property is a property that can have only one unique value y, for each instance x, i.e. there cannot be two distinct values y_1 and y_2 such that the pairs (x, y_1) and (x, y_2) are both instances of this property. Thus, in practice, non-functional properties can have multiple values.

In addition to functionality directly offered by the enums themselves, there is also a service class `StrainInfo2Ontology`[12] which offers additional functionality based on indices generated over all `Terms` and on an external XML resource. Upon construction, `StrainInfo2Ontology` generates indices to convert URIs (fully qualified names) or prefixed terms (used in some situations such as CSV headers) into `Terms`. In addition, it creates the inverse relation of domain and range

[10] `java.lang.Class`
[11] `http://www.w3.org/TR/owl-ref/#FunctionalProperty-def`
[12] `net.straininfo2.ontology.StrainInfo2Ontology`

(`getInDomainOf(Entity)` and `getInRangeOf(Entity)`). Information added through the custom XML file (`ontology.xml`), which is parsed during construction of the class, applies to both types of terms and contains information that is too bulky to be added to the enum files themselves. This is mostly textual information, such as the full HTML formatted definition and the short (one line) description used in the online reference. This mechanism is also used to add mapping information, the 'since' version, the unit (if applicable) and whether the property is deprecated. Controlled vocabularies are also defined in this file, and are called `value sets`. There are independent `value sets` which are explicitly named, and anonymous value sets, bound to the property they are defined for. Value sets are sets of `values`, which have a canonical form (`can`), alternatives (`alt`), and a description (`desc`). The canonical forms are the preferred values to be used when saving MCL data. Alternatives might occur in other resources and are listed to aid users in mapping. When the canonical value is a meaningless identifier such as a URI, the first alternative is considered the human readable version of the `value`. The description is a comment field providing an explanation that is included in the MCL reference.

Mapping between managed entities and MCL terms is performed by `OntologyMapper`[13]. This class translates an entity object (such as a `Sample` or `Medium`) into a `Map` with `Propertys` as keys and the actual values as values and vice-versa. The resulting `Map` can then be conveniently used in XML or RDF generation code. The inverse mapping is used to convert MCL XML in the XML synchronization procedure.

[13]`net.straininfo2.ontology.OntologyMapper`

Listing 6.1: Term: common interface for MCL terms.

```
package net.straininfo2.ontology;
public interface Term {
  /**
   * Returns the name of the term, without namespace
   * (e.g. speciesName or Culture).
   */
  public String getTerm();

  /**
   * Returns the namespace of the term.
   */
  public Namespace getNamespace();

  /**
   * Shorthand for getNamespace().getJdom();
   */
  public org.jdom.Namespace getJdomNamespace();

  /**
   * Returns the name of the term, including a namespace
   * prefix. (e.g. mcl:speciesName of mcl:Culture)
   */
  public String getPrefixedTerm();

  /**
   * Gives the full name, including the full namespace.
   * (e.g. http://www.straininfo.net/ns/mcl/2.0/strainNumber)
   */
  public String getFullTerm();

}
```

Listing 6.2: Property: modelling MCL properties.

```
package net.straininfo2.ontology;
public enum Property implements Term {

  STRAIN_NUMBER (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_STRAIN_NUMBER,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},
      null, null, null, false),
  OTHER_STRAIN_NUMBER (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_OTHER_STRAIN_NUMBER,
      new Entity[]{ Entity.CULTURE},
      null, null, false, false),
  OTHER_STRAIN_NUMBERS (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_OTHER_STRAIN_NUMBERS,
      new Entity[]{ Entity.CULTURE},
      null, null, false, false),
(...)
  SPECIES_NAME (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_SPECIES_NAME,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},
      null, null, true, false),
  QUALIFIED_SPECIES_NAME (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_QUALIFIED_SPECIES_NAME,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},
      null, null, true, false),
(...)
  TYPE_STRAIN_OF (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_TYPE_STRAIN_OF,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},
      null, null, true, null),
  TYPE_STRAIN_OF_SPECIES (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_TYPE_STRAIN_OF_SPECIES,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},
      null, Property.TYPE_STRAIN_OF, true, null),
  TYPE_STRAIN_OF_GENUS (Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_TYPE_STRAIN_OF_GENUS,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},
      null, Property.TYPE_STRAIN_OF, true, null),
(...)
  OPTIMAL_GROWTH_TEMPERATURE(Namespace.MCL,
      OntologyConstants.PROPERTY_MCL_OPTIMAL_GROWTH_TEMPERATURE,
      new Entity[]{ Entity.CULTURE, Entity.STRAIN},null,
        Property.GROWTH_TEMPERATURE, true, true),
(...)

  private final Namespace namespace;
private final String term;

private final Entity[] domain;
private final Entity[] range;
private final Property subPropertyOf;

private final Boolean isFunctional;
private final Boolean hasTripleRepresentation;

Property(Namespace namespace, String term, Entity[] domain,
    Entity[] range, Property subPropertyOf, Boolean isFunctional,
    Boolean hasTripleRepresentation ) {
    (...)
}

/*
 * Term implementation
 */

public String getTerm() {
  return term;
}
public Namespace getNamespace() {
  return namespace;
}
public org.jdom.Namespace getJdomNamespace() {
  return namespace.getJdom();
}
public String getPrefixedTerm() {
  return namespace.getPrefix() + ":" + getTerm();
}
public String getFullTerm() {
  return namespace.getURI() + getTerm();
}

/*
 * Other functionality
 */

public Boolean hasTripleRepresentation() {
  return hasTripleRepresentation;
}

public Entity[] getDomain() {
  return domain;
}
```

```
  public Entity[] getRange() {
    return range;
  }

  public boolean isFunctional() {
    if (isFunctional != null)
      return isFunctional;
    else
      return false;

  }

  public Property getParentProperty() {
    return subPropertyOf;
  }

  public boolean isPublicationRelation() {
    return this == PUBLICATION
      || this.getParentProperty() == PUBLICATION;
  }

  public boolean hasLiteralValue() {
    return range == null;
  }

}
```

This chapter is based on:

B. Verslyppe, R. Kottmann, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. Microbiological Common Language (MCL): a standard for electronic information exchange in the Microbial Commons. *Research in Microbiology*, 161(6):439–445, 2010.

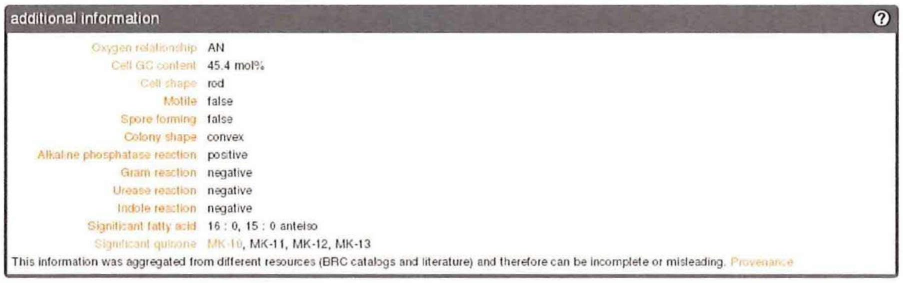
This publication results from a collaboration with Dr. Renzo Kottmann (Microbial Genomics Group, MPI for Marine Microbiology, Bremen, Germany) in the context of the Genomic Standards Consortium (October 2009).

We want to thank the MCL early adopters who have made their catalog available in XML and provided feedback on the language structure.

7 Semastri

StrainInfo has reached the point where it can be seen as a mature platform. There is a tremendous index containing cultures integrated into strains, and the web application has shown its capability for use in production. The integration result has been curated based on an error detection algorithm [37] and Histris which further improve the integration confidence (see Chapter 4).

Further growth of StrainInfo is possible in essentially four orthogonal directions. First, the index itself still has potential to grow as a long tail of BRCs remains to be indexed and readily indexed BRCs continuously assign new strain numbers. Second, the equivalence integration confidence can be further improved by finding new provenance that reconfirms links between strain numbers. Third, adding more functionality to the web application, such as more complex search queries or novel visualisations of the index. Fourth, making StrainInfo more information rich by adding more fine-grained information to the passports. The first and second direction are addressed with MCL XML synchronisation and Histri. The third is possible as the result of specific case studies. Although users also appreciate more coverage or more complex provenance, a larger, broader audience can be reached with practical, easy to understand information. Therefore, this chapter focuses on


additional information

Oxygen relationship	AN
Cell GC content	45.4 mol%
Cell shape	rod
Motile	false
Spore forming	false
Colony shape	convex
Alkaline phosphatase reaction	positive
Gram reaction	negative
Urease reaction	negative
Indole reaction	negative
Significant fatty acid	16 : 0, 15 : 0 anteiso
Significant quinone	MK-10, MK-11, MK-12, MK-13

This information was aggregated from different resources (BRC catalogs and literature) and therefore can be incomplete or misleading. Provenance

Figure 7.1: Semantic information gathered for the *Prevotella pleuritidis* type strain (cultures include `CCUG 54350`^{T}, `DSM 18744`^{T} and `JCM 14110`^{T}).

the last growth direction.

Semastri, short for semantic StrainInfo (and pronounced with a pun to 'chemistry'), is the project that corresponds to collecting, semantically integrating and presenting fine-grained information in StrainInfo. It brings StrainInfo past its initial purpose of identifier integration. An example of the outcome of Semastri is shown in Figure 7.1. This particular example is taken from the strain passport of the *Prevotella pleuritidis* type strain. The module is shown as the third module of the strain passport (below the Histri module) and lists information which otherwise is only available from BRCs (catalogs or internal information) or scientific literature. The fields correspond to MCL terms, and are linked to their definition in the MCL reference. This kind of practical information opens StrainInfo's scope to a broader audience, i.e. researchers interested in the properties of strains or species, but who are not necessarily working with BRC cultures. Semastri is currently only available in internal development versions of StrainInfo.

7.1 Integrating semantic information into StrainInfo

7.1.1 Overview

StrainInfo indexes cultures and their attached meta-information. This corresponds to the abstract concept 'culture resource': a resource about a culture, containing meta-information about that culture. The archetype culture resource is the BRC

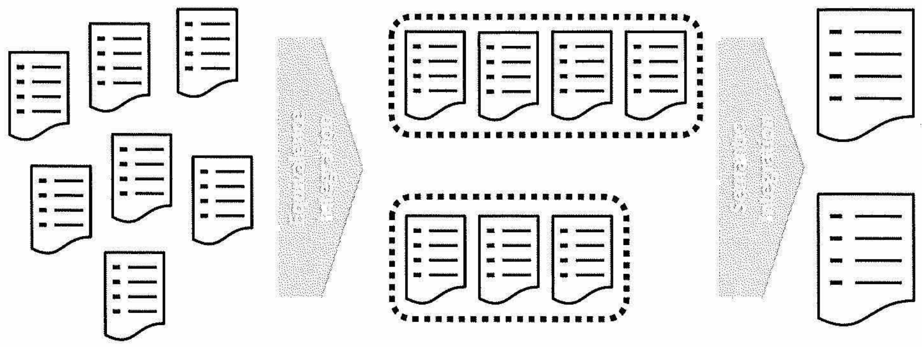


Figure 7.2: Equivalence integration clusters equivalent resources together, semantic integration combines all information into one resource. Resources contain key-value pairs with well-defined keys (e.g. MCL terms).

catalog entry. BRC catalog entries contain meta-information specifically tailored to the culture, structured as key-value pairs, with keys corresponding to known terms. Although culture resources give information about a specific linked culture, it is not necessary that the information source is inherently associated with precisely one culture. If a resource (such as a taxonomic resource) names multiple cultures in one record, this information can be seen as multiple identical culture resources bound to the different cultures.

Information enters StrainInfo at the culture level, in the abstract form of culture resources. As cultures are the only permanent fixture, strain passports must be seen as a derived view of that information. As discussed in Section 2.2.1, the integration process prior to the strain passport consists of two integration steps, i.e. equivalence integration and semantic integration (see Figure 7.2). The equivalence integration introduces the strain concept (which forms the basis for the strain passports). In fact, equivalence integration boils down to clustering culture resources, based on a range of clues (mostly `mcl:otherStrainNumber` fields), which are (at least theoretically speaking) part of the culture resources.

To make the clustered culture resources useful, they are semantically integrated to be able to generate strain passports. This amounts to taking all information from those culture resources, combining information, removing redundancy and saving the result as a 'strain resource'. Strain resources are conceptually identical to culture resources, except that they model resources that are one 'equivalence

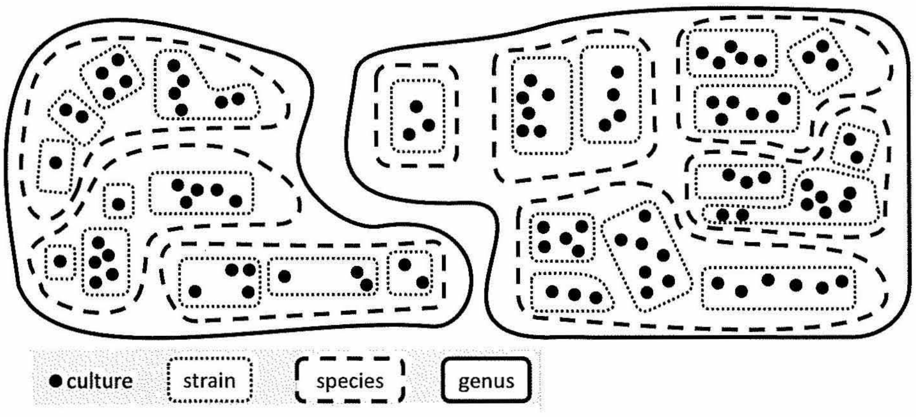


Figure 7.3: Equivalence integration creates clusters, while semantic integration unifies the clusters. Each cluster therefore can also be seen as a resource and subsequently clustered, yielding a topology of clusters.

integration level' higher than culture resources. In addition, the semantic integration is related to the equivalence clustering: an incorrect equivalence clustering will yield information bits that are hard or impossible to semantically integrate.

Strain resources can also be clustered. The natural way is to use the species name on the strain resource as the criterion for strain equivalence clustering (or strain equivalence integration). The species name listed on a strain resource is not necessarily the one that is stated on the underlying culture resources, as semantic integration of the species name at the culture level uses majority voting to select the most probable species name. The integration result is a species resource, which contains information based on all strains (and therefore only indirectly all cultures) that belong to the species. The information can be presented on the corresponding taxon passport. In addition, species resources can be subsequently clustered, based on the genus the species belong to, and semantically integrated, to form genus resources. This leads to a topology as shown in Figure 7.3, where each culture is a representative of a strain, species and genus. In practice, exceptions to this situation exist: strains can for example only be classified at the genus level. Furthermore, each type of resource also corresponds to and populates a passport on StrainInfo. Culture resources correspond to BRC catalog entries or other information sources.

7.1.2 Semantic information sources

Semantic information in StrainInfo comes from many different sources, and can be seen as a collection of culture resources. The Semastri pipeline is shown in Figure 7.4. Essentially, information is extracted in culture resource form, and represented as RDF triples, which are fed to the semantic integration algorithm. The intermediate representation as RDF triples is necessary to make the wide range of data sources uniform, and as a data model for the semantic integration algorithm. The data sources currently used include: stored legacy information from BRC catalog crawling, Histri data, MCL XML synchronisation files and information extracted from literature. The pipeline can be extended easily to include user generated information (in the sense of Web 2.0; ignoring the fact that Make Histri data is already user generated) or other information sources by transforming their culture resources into RDF triples representation.

Information accessible from the database

MCL XML synchronisation brings a large amount of semantic information in an electronic format to StrainInfo. This is a good source of semantic information: it is well-structured (as XML files), relatively information rich (all fields are marked with MCL terms) and it is available on a large scale (major BRCs perform MCL XML synchronisation). However, this information is also limited to what BRCs have in their catalogs and what they are willing to share. There is a suggested set of MCL terms that can be included in the MCL XML (catalog schema) files, but the focus of those terms lies on information relevant for the equivalence integration and Histri-related fields. The XML CLOBs are converted into RDF representation, which is a relatively straightforward task as the data is already marked with MCL terms and linked to a culture[1].

The Make Histri project also generates information that is used by Semastri. First, there are the Histri trees themselves, but these must actually be seen as the semantic integration result of the textual history descriptions (corresponding to the MCL term `mcl:history`). Strictly speaking, the reconstructed Histri tree itself does not add information to the existing culture resources, but realises a higher level view (i.e. at the strain level). Therefore, the Histri tree belongs to the resulting strain resource. However, users are also able to add information at the strain level (using

[1]The RDF representation is constructed from the `CultureTriple` representation in the relational database. `CultureTriples` are generated during the integration of the MCL XML files.

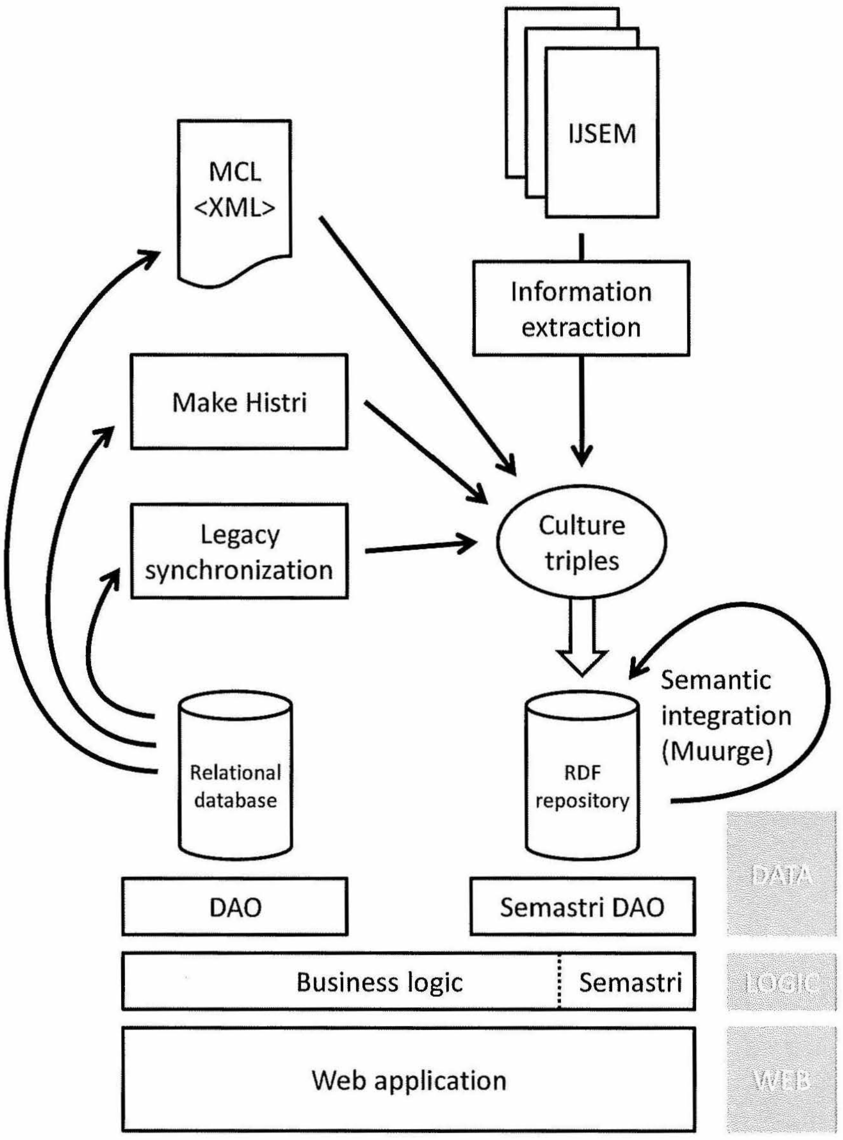


Figure 7.4: Semastri pipeline and relation to the existing StrainInfo platform architecture.

the strain tab). This includes the isolation date, isolator and his affiliation, the sample habitat description and sample location. This information is bound to the marked isolate, as this is the culture to which this information assuredly applies. If there is no marked isolate, an arbitrary reference culture is chosen.

The legacy database also contains information that can be used by Semastri. The species names stored in `Culture` records are in fact also part of the culture resources. This is also the case for the crawls of taxonomic databases, which contain information about type strain status and species names, and which can be bound to the cultures listed on those entries. In addition, some very diverse meta-information stored on old records part of the legacy screen scraping procedure are converted to RDF triples.

Information extracted from scientific literature

The challenge that follows logically, is the question where better semantic information can be found, overcoming the coverage (in terms of information richness) limitations that the already existing sources of semantic information exhibit. A natural information source is the vast body of scientific literature produced by the microbiological community. Unfortunately, the semantic information facts themselves are scattered over many articles and textbooks, buried in sentences and trapped by with English grammar. This is problematic for computer algorithms which require structured electronic data formats. Therefore, in an attempt to release that information, we started an Information Extraction (IE) experiment.

Essential is the selection of a corpus of relevant documents from which the information will be extracted. The *International Journal of Systematic and Evolutionary Microbiology* (IJSEM), published by the Society for General Microbiology (SGM), was chosen as this is the central journal containing rigorous species descriptions, described in a very specific and uniform manner. The publications were collected by crawling the HTML full-text articles online. Before facts can be linked to organisms, the cultures to which those facts will be linked need to be identified. Therefore, named entity recognition (NER) of strain numbers and taxonomic names is performed using a dictionary approach. The strain numbers are used to link the extracted facts to the actual microbial material (and as a side effect also improve the StrainInfo literature index), while the taxonomic names are used as additional contextual information and text boundary markers. Facts are extracted by matching frequently occurring patterns and mapping recommended values of MCL terms, combined with other clues in the same sentence. After post-processing, facts are represented as RDF triples to be semantically integrated.

This corpus is processed using a pipeline that makes use of the Apache UIMA (Unstructured Information Management)[2] framework. Apache UIMA is an Apache-licensed open source implementation of the UIMA specification. It enables applications to be decomposed into components that implement interfaces defined by the framework and provide self-describing metadata via XML descriptor files. These components do the actual work of analyzing unstructured information, by adding annotations to the data. Users can write their own annotators, or configure and reuse pre-existing annotators that are available as part of the Apache project or from third parties. Most annotator components add annotations to the UIMA data model, based on other annotations or new elements selected in the text. Annotations comprise of a text snippet (called a 'span'), and optional fields storing arbitrary information that can be utilised by subsequent annotators. The framework manages the components and the data flow between them. Components can be written in both Java and C++. The data that flows between the components is designed for efficient mapping between both languages.

The pipeline consists of many relatively small annotator components. It starts with stripping the HTML and storing the plain text in a separate aspect of the documents, which is later used by most annotator components. Next, the bibliographic details of the documents in the corpus are recovered from the HTML headers stored in the crawled files. This information is later used for provenance. Subsequently, analysis of the plain text is started by tokenizing the text. Tokenization is the process of splitting text into words, which are called tokens. The type strain 'T' suffix is removed from the tokens as it interferes with `ConceptMapper`, a scalable and highly configurable dictionary annotator component of the Apache UIMA project. It is used to annotate strain numbers and species names with their cultureId and taxon id respectively. To improve recall, the dictionary also includes species names with an abbreviated genus epithet (if unique). In addition, values from MCL recommended vocabularies are also annotated. INSDC sequence accession numbers are annotated using a regular expression. The subsequent annotator annotates special frequently occurring artifacts (`StrainArtifacts`) such as `sp. nov.` and `Description of`, which are used as boundaries. Using these artifacts and the taxon annotation of the `ConceptMapper`, the species description section titles are annotated. Once the section titles are known, the whole section is annotated, and the typical three paragraphs (etymology, description and type strain information) are detected. Only the

[2]`http://uima.apache.org/`

second paragraph contains multiple facts and therefore it is split into sentences using the GENIA sentence splitter (GeniaSS, [76]). Subsequently, these sentences are scanned independently for facts. `SelectKVCandidatesAnnotationEngine` generates `KVCandidate` annotations, which are candidate key-value pairs for culture resources. They are generated from the sentences based on pattern matching (mostly numbers and units) and the MCL recommended vocabulary annotations. After the generation phase, there is a heuristic post-processing phase (`KVCandidatesPostProcessingAnnotationEngine`) that selects the most probable candidate if multiple conflicting `KVCandidates` are found (e.g. multiple values for the same non-functional MCL term). In addition, the annotator also selects the right MCL term if there are multiple options (e.g. `mcl:noGrowthTemperature`, `mcl:minimalGrowthTemperature`, `mcl:optimalGrowthTemperature` and `mcl:maximalGrowthTemperature`). The `KVCandidate` annotations are the end result to be represented in RDF triples. They are matched with the culture annotations of the species description section to populate output culture resources. The results of the UIMA structure are written to multiple output files. The most important one is the RDF representation of the KVCandidates. The strain number and taxonomic name dictionary annotations are also saved as they can be used to improve the literature index of StrainInfo, together with other annotations for a variety of uses.

MCL was extended with new terms and controlled vocabularies to model the richness of the semantic information in the species description sections. These additions were necessary as the creation of `KVCandidate` annotations depends on the annotated values from MCL controlled vocabularies found by `ConceptMapper`. This extension corresponds to MCL version 2.1.

7.1.3 Semantic integration algorithm

The RDF representation is loaded into an RDF repository. We have chosen OWLIM-SE[3], as it is scalable (being able to handle billions of triples), and interfaces using OpenRDF Sesame[4]. The semantic integration algorithm iteratively interfaces with the repository to extract input resources and to store results of the integration round.

The semantic integration algorithm is a composition of rules that generate a consensus value for each term occurring in the set of culture resources (see Fig-

[3]`http://www.ontotext.com`

[4]`http://www.openrdf.org`

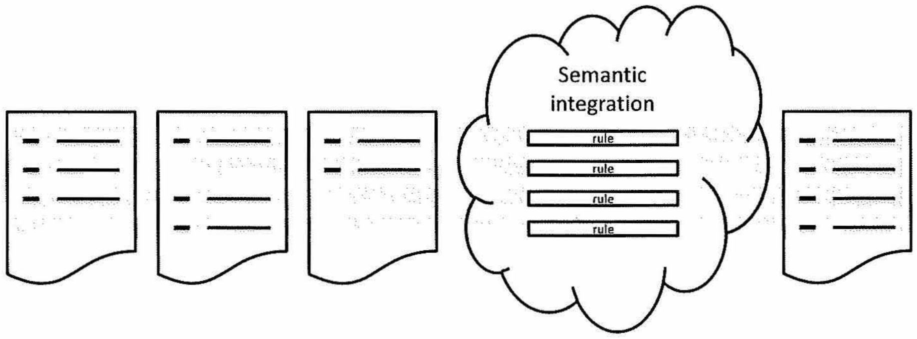


Figure 7.5: The semantic integration algorithm is a composition of rules that semantically integrate the corresponding fields (displayed at the same height) on the resources.

ure 7.5). Each MCL term is handled using a heuristic tailored to the semantics of the term. The semantic integration algorithm is created based on the Muurge framework (see Chapter 8), which allows to set up custom rules for input resources and to reuse standardized integration heuristics. If a term can not be semantically integrated, an exception is generated and stored in the integration result. The main integration heuristics used in the semantic integration of culture resources are:

Equality. Values must be equal or integration is not possible (i.e. yields an exception). This heuristic is used for terms that can only have a single value for all resources. An example is the (actually virtual) field that contains the reference to the strain, species or genus cluster the resource belongs to (i.e. the equivalence integration result must be identical as otherwise the algorithm is semantically integrating non-equivalent resources). In practice, majority voting is preferred over this heuristic as it is able to handle inconsistencies.

Majority vote. Select value with highest frequency. Optionally yields an exception if the voted value has an occurrence less than a given minimal threshold. It can be used to cope with inconsistencies: when the disagreement occurs in only a minority of the resources (i.e. a small disagreement between values given by different resources), it is neglected.

Examples: `mcl:speciesName`, `mcl:isolationDate`

Aggregate to set. Collect all different values, removing duplicates in the process.

This is often used for non-functional MCL terms, which can have multiple independent statements with the same term. This heuristic then collects all possible values as given by all resources combined.

Examples: `mcl:strainNumber`, `mcl:inhibitedBy`, `mcl:utilizes`

Enclosing range. Determine the smallest enclosing range that fits all numbers. This is used for values where some variability (e.g. on measurements) is possible.

Examples: `mcl:growthTemperature`, `mcl:optimalGrowthNaCL`

Ontology reasoning. Use the knowledge of an underlying ontology to determine an integration result. In hierarchical ontologies, this can including finding the first common ancestor or retaining all leaves.

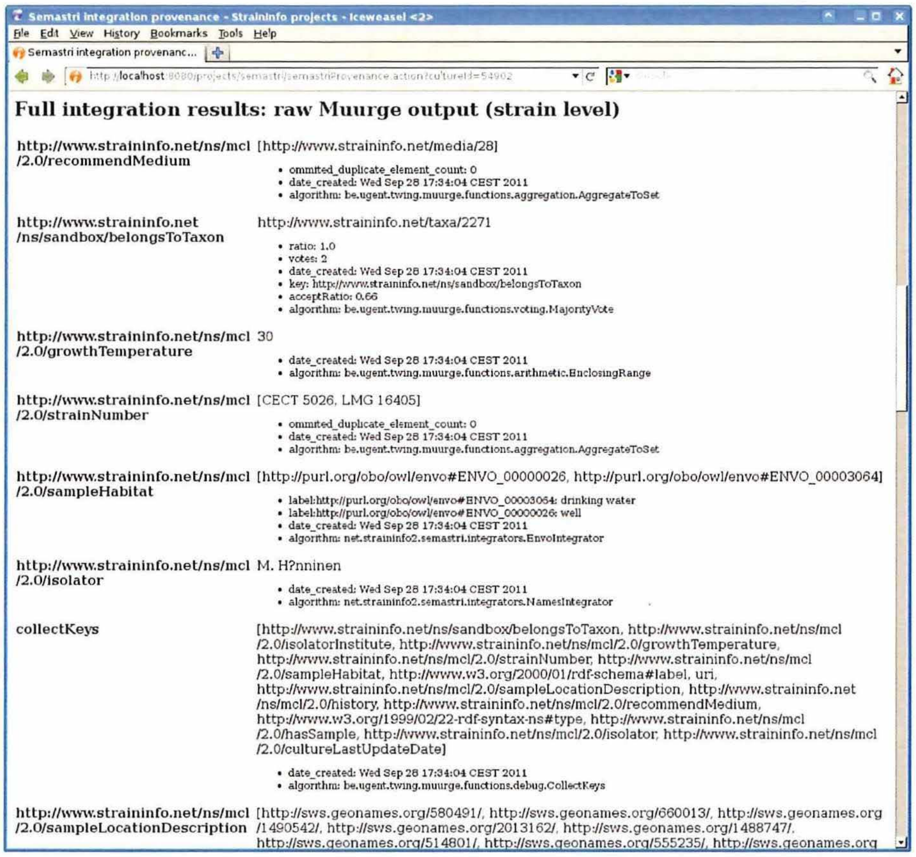
Examples: `mcl:sampleLocationDescription`, `mcl:sampleHabitat`

The configuration of the rule composition depends on the semantic integration goal. There is a slight conceptual difference between intra-strain (i.e cultures to strain) and inter-strain (i.e. strains to species and species to genus) integration. The first assumes that values must be identical as they describe equivalent organisms, while the latter has a 'collecting' nature. For example, the `mcl:isolator` of all cultures belonging to a strain must be the same name, while for a set of strains this might be a set of different people. This results in two semantic integration algorithms being used on three integration levels.

7.2 Discussion

Semantic information from Semastri is available on passport pages and a separate project homepage, similar to the Make Histri project. The Semastri project homepage contains links to some semantic integration and provenance examples. All integration results have a dedicated provenance page, which contains the full output of the semantic integration algorithm. An example is shown in Figure 7.6. In addition, there is experimental search functionality and a feature to compare strains by all given values for a term. This can be used to compare species (or other sets of strains), and is especially suited for comparing the values of non-functional terms.

For the information extraction experiment, we crawled 1972 publications, starting from volume 53 (2003), where HTML full text availability starts. The collected


Semastri integration provenance - StrainInfo projects - Iceweasel <2>
File Edit View History Bookmarks Tools Help
Semastri integration provenanc...
http://localhost:8080/projects/semastri/semastriProvenance.action?cultureId=54902

Full integration results: raw Muurge output (strain level)

http://www.straininfo.net/ns/mcl/2.0/recommendMedium [http://www.straininfo.net/media/28]

- ommited_duplicate_element_count: 0
- date_created: Wed Sep 28 17:34:04 CEST 2011
- algorithm: be.ugent.twing.muurge.functions.aggregation.AggregateToSet

http://www.straininfo.net/ns/sandbox/belongsToTaxon http://www.straininfo.net/taxa/2271

- ratio: 1.0
- votes: 2
- date_created: Wed Sep 28 17:34:04 CEST 2011
- key: http://www.straininfo.net/ns/sandbox/belongsToTaxon
- acceptRatio: 0.66
- algorithm: be.ugent.twing.muurge.functions.voting.MajorityVote

http://www.straininfo.net/ns/mcl/2.0/growthTemperature 30

- date_created: Wed Sep 28 17:34:04 CEST 2011
- algorithm: be.ugent.twing.muurge.functions.arithmetic.EnclosingRange

http://www.straininfo.net/ns/mcl/2.0/strainNumber [CECT 5026, LMG 16405]

- ommited_duplicate_element_count: 0
- date_created: Wed Sep 28 17:34:04 CEST 2011
- algorithm: be.ugent.twing.muurge.functions.aggregation.AggregateToSet

http://www.straininfo.net/ns/mcl/2.0/sampleHabitat [http://purl.org/obo/owl/envo#ENVO_00000026, http://purl.org/obo/owl/envo#ENVO_00003064]

- label:http://purl.org/obo/owl/envo#ENVO_00003064: drinking water
- label:http://purl.org/obo/owl/envo#ENVO_00000026: well
- date_created: Wed Sep 28 17:34:04 CEST 2011
- algorithm: net.straininfo2.semastri.integrators.EnvoIntegrator

http://www.straininfo.net/ns/mcl/2.0/isolator M. H?nninen

- date_created: Wed Sep 28 17:34:04 CEST 2011
- algorithm: net.straininfo2.semastri.integrators.NamesIntegrator

collectKeys [http://www.straininfo.net/ns/sandbox/belongsToTaxon, http://www.straininfo.net/ns/mcl/2.0/isolatorInstitute, http://www.straininfo.net/ns/mcl/2.0/growthTemperature, http://www.straininfo.net/ns/mcl/2.0/strainNumber, http://www.straininfo.net/ns/mcl/2.0/sampleHabitat, http://www.w3.org/2000/01/rdf-schema#label, uri, http://www.straininfo.net/ns/mcl/2.0/sampleLocationDescription, http://www.straininfo.net/ns/mcl/2.0/history, http://www.straininfo.net/ns/mcl/2.0/recommendMedium, http://www.w3.org/1999/02/22-rdf-syntax-ns#type, http://www.straininfo.net/ns/mcl/2.0/hasSample, http://www.straininfo.net/ns/mcl/2.0/isolator, http://www.straininfo.net/ns/mcl/2.0/cultureLastUpdateDate]

- date_created: Wed Sep 28 17:34:04 CEST 2011
- algorithm: be.ugent.twing.muurge.functions.debug.CollectKeys

http://www.straininfo.net/ns/mcl/2.0/sampleLocationDescription [http://sws.geonames.org/580491/, http://sws.geonames.org/660013/, http://sws.geonames.org/1490542/, http://sws.geonames.org/2013162/, http://sws.geonames.org/1488747/, http://sws.geonames.org/514801/, http://sws.geonames.org/555235/, http://sws.geonames.org

Figure 7.6: Provenance of the semantic integration (strain level).

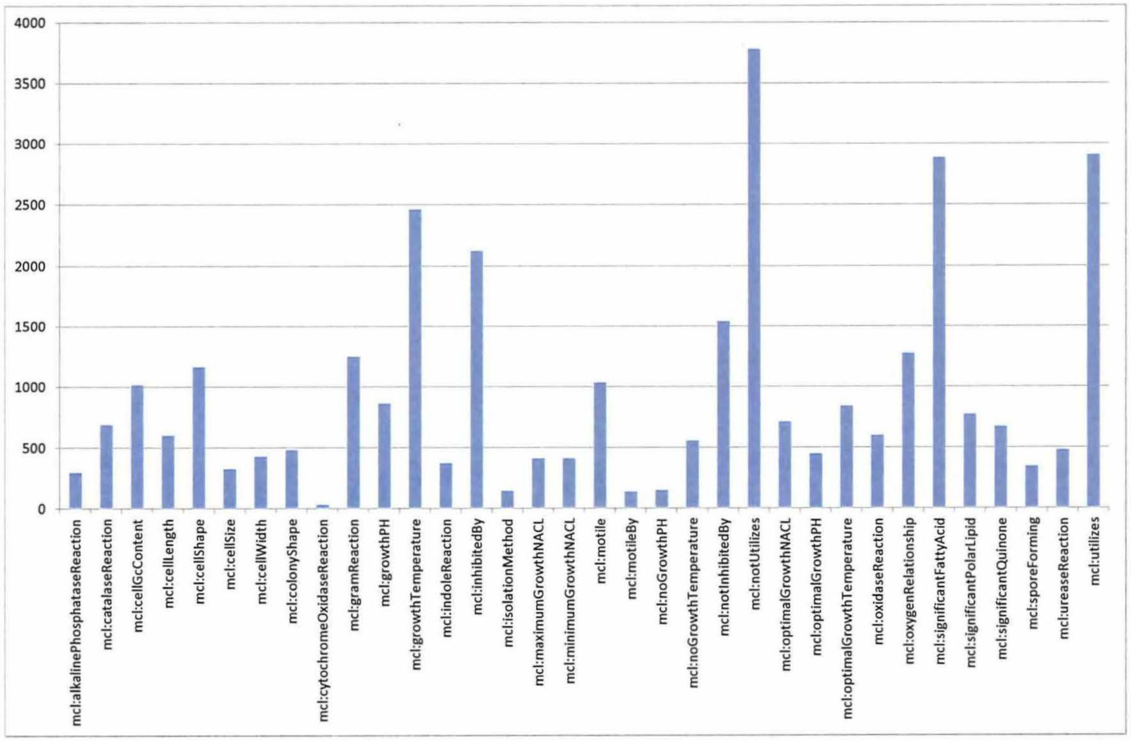


Figure 7.7: Number of facts extracted from the IJSEM corpus per MCL term.

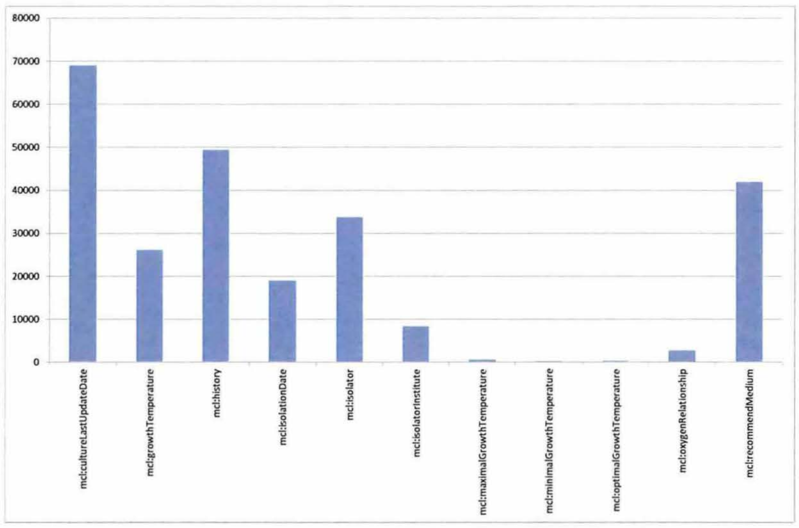


Figure 7.8: Number of facts extracted from BRC catalogs (MCL XML) per MCL term.

corpus corresponds to about half the total availability of HTML full text publications (which later became inaccessible due to content protection installed on the IJSEM web server). Older publications are only available as PDF files that need Optical Character Recognition (OCR) to extract the plain text. From the publications in the corpus, we extracted 2007 description sections; of which 1545 describing new species, 335 describing new genera and 127 novel combinations. On average, a description section contains 12.11 sentences in its description paragraph, which contain on average 16.04 statements per paragraph (or 1.32 statements per sentence), totalling up to 32 184 statements. The MCL term distribution of those facts is shown in Figure 7.7. Non-functional terms generally have a higher prevalence as for those terms often enumerations of different possible values occur. As can be seen from Figure 7.8, the term distribution is considerably different in terms of term coverage and number of facts from the MCL XML synchronisation and legacy crawl artifacts (the latter corresponding to 320 060 RDF statements).

In only 1280 description sections a 'The type strain is (...)' paragraph is discovered. This type of paragraph, always the third paragraph in a description section, contains the strain numbers of the type strain (necessary to be able to link the facts to cultures), and often contains information about the sample location and habitat. The annotator uses a couple of heuristically determined patterns to determine whether the third paragraph is such type of paragraph. If strain numbers are available and can be mapped to cultures, the extracted facts are linked to the cultures instead of the taxonomic name in the section title. This is because the experiments to determine the morphological and biochemical properties of the bacteria are performed on the type strains. In addition, this also avoids problems when taxa are renamed, as the facts stay coupled to the underlying strains. Nevertheless, if there are no cultures mentioned, the extracted facts are linked to the taxon named in the section title.

The information extraction experiment performed here must be seen as an intermediate solution towards the adoption of semantic microbiological information generated in suitable electronic representation formats. The richness of the extracted facts compared to the MCL XML and legacy BRC crawls justifies the effort required for information extraction, but a long-term vision would be to generate the information in an electronically processable semantic way from the start. In such a scenario, users would no longer write species descriptions in prose, but enter the information in a structured way, using MCL terms, for example using an application that assists in completing a form. For current publication purposes, the structured electronic format can still be converted to text, which can be done in an automated manner without the need of human interference. Microbiology would benefit from an authoritative source of such information, as technologically enforced richness and diversity of new species descriptions would overcome some of the problems of the current practice of publishing of new taxa.

An initial proof of concept of semantically integrating sample habitat and location using ontologies was built during an internship at the Knowledge Media institute (KMi) of The Open University in Milton Keynes, UK (December 2009), working with Prof. Enrico Motta, Ph.D. and Andriy Nikolov, Ph.D..

General semantic integration of microbiological facts extracted from literature has been worked out during a study leave at the USC Information Sciences Institute in Marina del Rey, Los Angeles, CA, USA (September 2010–July 2011), working with Prof. Eduard Hovy, Ph.D., Gully Burns, Ph.D. and Cartic Ramakrishnan, Ph.D..

8 Muurge: merging near-identical raw data

When two or more independent sources give exactly the same value for a field, the redundancy confirms the actual value and an increasing number of sources stating the value adds to the confidence. The human mind has a natural gift for synthesising facts or stories learned from different sources. For example, there are many opportunities to learn the name of the band that plays the song 'Nothing Else Matters'. Some people will learn this by actively listening to Metallica albums or hearing about this song from their parents or friends. In addition, every time this song is announced on the radio is also a chance to learn the name of the band that plays the song. When asked the band name, people can only respond with the right name if they had the chance to learn the correct answer. They will give one solution synthesised from all learning opportunities, and will not refer to all individual instances where they were confirmed that this band performs this song. Even if a radio voice would mistakenly announce this song as being played by Nirvana, the synthesising ability will enable humans to look at the overwhelming confidence of the relation with Metallica in order to determine that this particular fact is wrong.

In addition, the human mind also has the capability to naturally combine multiple incomplete stories into one consensus story, *fusing* or *merging* different data sources into one, consistent output.

Surprisingly, redundancy is the main problem that needs to be overcome when performing semantic integration. More specifically, the situation where two values are equivalent, but not syntactically identical and therefore seen as distinct values by computer algorithms is problematic as preferably only one value should be shown to the user. Two values can be syntactically different as the consequence of:

- Some values giving more detail than others (for example, `May 1990` vs. `1990` for an isolation date field or `P. De Vos` vs. `Paul De Vos` for an isolator field).
- Values with a different field granularity, combining several fields in one or distributing sub values into multiple fields (e.g. `P. De Vos, 1990`, containing both the isolator and the isolation date into one field vs. the two separate fields `P. De Vos` and `1990`).
- Syntactical differences; values using other notations to indicate the same (e.g. values in different units).
- Values in multiple languages. This can be the case for textual descriptions, but also for names such as geographical location names (e.g. `Gent` vs. `Ghent` vs. `Gand`).

We believe this touches upon one of the fundamental problems that still needs to be solved by computer science (but where a large research effort is focused on). We have come to the point where we are filling a world-wide system (the Internet) with data we collect from all different sources imaginable. However, we lack the technology to adequately keep an overview of or summarise that data using a mechanism similar to our natural ability. The Semantic Web as envisioned by Berners-Lee [77] promised that it would solve this problem, and although it resolves data interoperability issues, it does not acquire an overview of the data. For example, in [78] Finin collects personal information from FOAF (Friend Of A Friend) files from the Semantic Web and aggregates information found in different files. In a particular example given in this paper, two different names are found for one person. Finin already states that merging is necessary, but should be performed with care as some facts may be wrong and a collection of facts may even contain contradictions.

This chapter only discusses semantic integration, which provides a *consensus value* for each field. The problem of clustering equivalent resources (also sometimes referred to as *instance coreference resolution*) is assumed to be solved. The semantic integration process is far less trivial as it might seem at first sight. Each field needs a dedicated algorithm, specifically tailored based on the format of the data and semantics of the field. However, many of those algorithms are related to each other or are variations on a common theme. Therefore, the integration algorithms can be filed in broad classes.

As this problem occurs in a wide range of different contexts, the solution was developed as a separate framework called 'Muurge'. The users of this framework are the developers using the framework to perform semantic integration tasks. The real end-users do not need to understand that the Muurge framework is being used. The following requirements were considered when designing the framework architecture:

- Lightweight and easy to use. Prefer convention over configuration and use smart defaults so that it is fast to set up but has extensive fine-tuning controls.
- Flexible. It needs to be usable in many different situations and configurations, and applicable to many different data sets.
- Make the framework elegant, easy to understand and fast.
- Extendable. If it does not solve a problem in a specific context well enough, make sure it is easy to be extended by the users.

8.1 Architecture

8.1.1 Overview

The high-level workflow of the Muurge framework is shown in Figure 7.5. The framework takes input `Resources`, performs semantic integration, and returns an output `Resource`. The heavy lifting is performed by `Muurgers`: they contain the rules, provide the execution context, connect the input to the Muurge rules and output a single end result.

`Resource`[1] is the data structure that is used for input and output. It is a generalisation of the data structure discussed in Chapter 7. `Resources` consist of

[1] `be.ugent.twing.muurge.Resource`

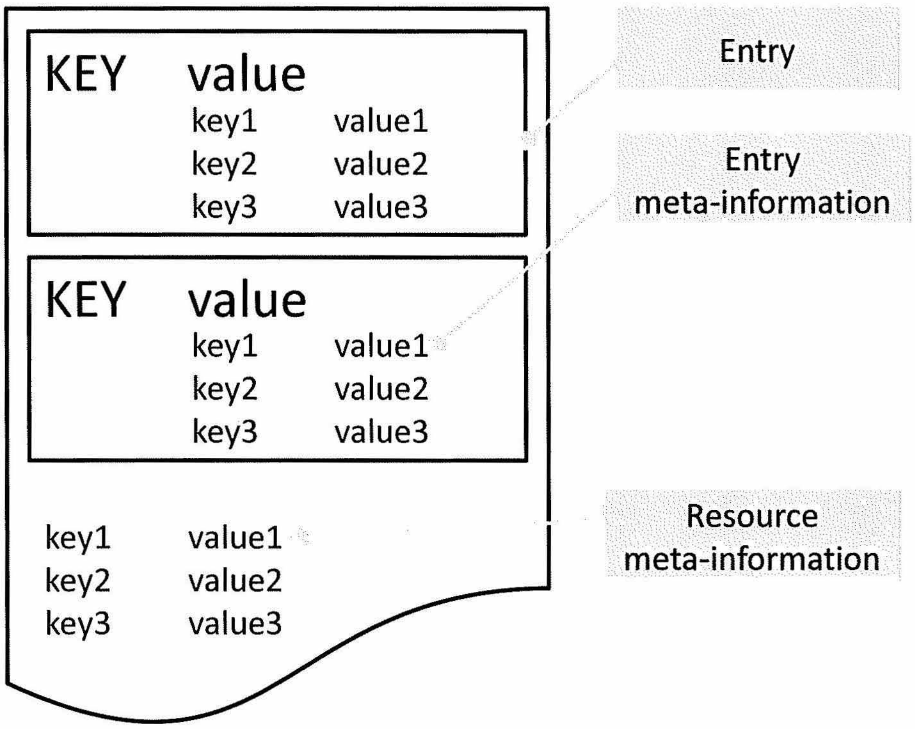


Figure 8.1: Muurge Resources consist of entries (key-value pairs). Entries as well as the Resource itself can have meta-information annotations (which are also key-value pairs).

entries, as shown in Figure 8.1. Each entry contains a key that can be used to retrieve the entry, and an entry value, for which a consensus value needs to be calculated. In addition, all entries can be annotated with meta-information (in the form of key-value pairs). There is also global meta-information on the Resource itself. This meta-information can be used for identifying or tracking the Resource.

As Resource is an interface, the framework needs actual implementations to interact with. The framework delivers a standard implementation based on default Java data structures with the DefaultResource[2] class. This implementation is highly compatible with the declarative configuration of the framework. The ad-

[2]be.ugent.twing.muurge.resources.DefaultResource

vantage of loose coupling with the actual implementation is that the interface can be added to existing data structures of other applications, so that they can be handled seamlessly by the Muurge framework. It also becomes possible to back `Resources` by other data representation technologies, such as a JSON (JavaScript Object Notation[3]), RDF or even a database.

The structure of a `Muurger`[4] is shown in Figure 8.2. The `Muurger` base class contains `MuurgeRules`[5], and is also responsible for their execution. Conceptually, a `MuurgeRule` performs a complete merge operation for one entry. Based on configuration, it selects the entries it applies to from all input `Resources`, calculates a consensus value, and outputs that value as an entry in an output `Resource`. `MuurgeRules` consist of `Conditions`[6], which perform pre-integration checks and a `Function`[7], which performs the actual integration. All components can be individually configured, and the configuration is referred to as the context. Contexts are hierarchically structured: contexts inherit from surrounding contexts. If a configuration property is not found, it is resolved from the configuration of the surrounding parent. Contexts are used to configure conditions and functions, such as a threshold or sensitivity setting on a condition, a configuration flag on the function or the key of the entry the conditions and function operate on. As the latter is generally shared by all `Conditions` and the `Function` of a rule, this is generally configured on the `MuurgeRule` level. In addition, all components also assume convention above configuration. Therefore, with a minimal number of configuration parameters chosen on strategic levels, the framework can be completely configured in an easy way.

If the input entries have an inconsistent or conflicting nature, it can be impossible to determine a consensus value. This is an exception state: it was impossible to determine a consensus value. In this case, `MuurgeRules` return an empty (`null`) entry in the output resource. The entry's meta-information is used to document the exception. `MuurgeRules` check all conditions first, and only apply the Muurge function if they succeed. Therefore, `Conditions` must be seen as reusable input validators. Next to exceptions, there are also errors. This happens when something goes wrong in the Muurge rule itself or in the framework. In this case, there is no output entry, but the java `Exception`[8] will be caught by the framework and logged.

[3] `http://www.json.org/`
[4] `be.ugent.twing.muurge.Muurger`
[5] `be.ugent.twing.muurge.MuurgeRule`
[6] `be.ugent.twing.muurge.Condition`
[7] `be.ugent.twing.muurge.Function`
[8] `java.lang.Exception`

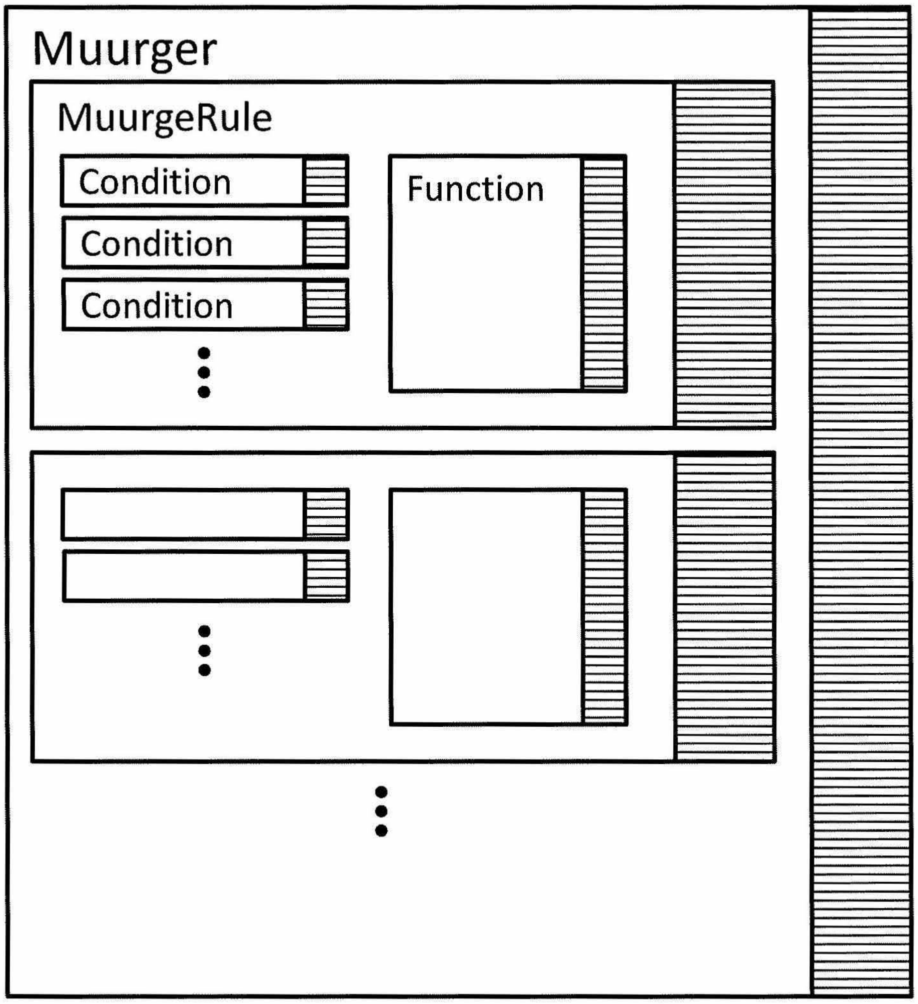


Figure 8.2: `Muurgers` perform the algorithmic tasks in the Muurge framework and contain the actual semantic integrators (`Functions`). The shaded borders represent the configuration context of the objects.

8.1.2 Conditions and functions

Conditions act as reusable input validators that guard Functions from executing on incorrect input. All conditions must be met in the MuurgeRule before the Function is executed. If one condition fails, the MuurgeRule is not further executed, and therefore yields an empty output Resource. Checks include making sure that entries exist, making sure that they are different from null and guaranteeing that the entry values are of a certain data type. Note that failing Conditions are different from inconsistencies found by Functions. The latter results in a null output entry which contains information about the exception in the attached metadata.

There are two broad classes of Functions: functions that output a single output value and Functions that output a collection (i.e. a list or a set) of multiple output values. Tables 8.1 and 8.2 give some examples of standard functions included (or planned) in Muurge. The behaviour of most implemented Functions can be modified via their configuration properties. In addition to the standard functions included in Muurge, two additional, customized functions were developed for the semantic strain integration in Semastri. They integrate the isolation sample location (mcl:sampleLocationDescription) and isolation sample habitat (mcl:sampleHabitat) fields respectively.

The mcl:sampleLocationDescription integration is implemented in GeoNamesIntegrator[9]. The textual geographic location description is annotated with so-called features from the GeoNames[10] ontology. This yields a multitude of annotations, each annotation matching a name with one or more geographical features. As a large number of geographic names is not unique (e.g. Cambridge becoming annotated with both the USA and the UK instance), there is a high occurrence of different features annotated to the same name. To correct this, irrelevant annotations are removed by using other higher order features such as countries or continents found in the set of input resources. In addition, the most specific feature is selected by removing the higher order features as this is redundant information that can be inferred from the ontology. The remaining annotation is the integration result; multiple remaining annotations or features being too distant indicate inconsistent data.

The mcl:sampleHabitat entries are integrated using a similar algorithm im-

[9] net.straininfo2.semastri.integrators.GeoNamesIntegrator
[10] http://www.geonames.org/

True consensus value

Integration function	Description	Input	Output
Equality	All input values must be equal	Thing	Thing
Majority vote	Take input with highest frequency	Thing	Thing
Minority vote	Take input with lowest frequency	Thing	Thing
Select random	Take random input value	Thing	Thing
Select index	Take input value with given index	Thing	Thing
Average	Take average of all inputs	Number	Number
Min	Take min of all inputs	Number	Number
Max	Take max of all inputs	Number	Number
Longest common substring	Take longest common substring	String	String
First common parent	Find first common parent in ontology	Ont.Term	Ont.Term
Overlay	Superimpose images	Image	Image

Table 8.1: Muurge `Functions` with single output. Ont.Term stands for ontological term.

Aggregation of single input values

Integration function	Description	Input	Output
Aggregate to list	List all inputs in random order	Thing	Thing[]
Aggregate to ordered list	List all inputs, sort afterwards	Thing	Thing[]
Aggregate to set	List all inputs, remove duplicates	Thing	Thing{}
Remove parents	Retain most precise terms	Ont.Term	Ont.Term{}
Supported lineage	Retain lineage with most support	Ont.Term	Ont.Term[]
Range	Create best enclosing range, i.e. pair of min,max	Thing	Thing..Thing
Animation	Make stop motion animation	Image	Video

Aggregation of collections

Integration function	Description	Input	Output
Union to set	All elements in one or more input sets	Thing[]	Thing[]
Union to list	Aggregate all input collections in random order	Thing[]	Thing[]
Intersection	All elements in all inputs sets	Thing[]	Thing[]
Symmetric difference	All elements in one set, but not in other	Thing[]	Thing[]
Deduplicate sets of names	Aggregate names, using fuzzy matching to remove duplicates	String{}	String{}

Table 8.2: Muurge `Functions` with multiple outputs. Square brackets denote a list; curly braces denote a set.

plemented in EnvoIntegrator[11]. It integrates the EnvO[12] [79] terms mapped from the textual habitat descriptions. However, as having multiple annotations increases the information content and therefore does not indicate inconsistencies, the integrator only removes redundant higher order terms (parents of other terms).

8.1.3 Framework configuration

The framework can be configured in two distinct ways: programmatically and declaratively. All components are standard Java objects which can be constructed and combined in Java code. This way, complete Muurgers, with Muurge rules and corresponding configuration can be constructed programmatically. Using programmatical configuration has the advantage of flexibility at run time (i.e. the Muurge rules can be adapted to the input). However, it has the tendency to become bulky even though Muurgers automatically assume a default configuration, lowering the boilerplate overhead for properties that have a smart default setting.

As most Muurgers have a static nature (i.e. they are designed once and are reused many times), many users will give preference to the declarative configuration. The smart design of the components and the predefinition of standard rules makes configuration using the standard Spring Inversion of Control XML configuration easy. The predefined rules are defined using the prototype scope in Spring. They contain the Function they are named after, and also already contain suitable conditions such as KeyExists[13]. As shown in example Listing 8.1, the parent property is used to insert a clone of a prototype rule, after which a custom configuration is added.

Listing 8.1: Example declarative configuration of a Muurger. This Muurger can be used to semantically integrate culture resources.

```
<bean id="exampleMuurger" class="be.ugent.twing.muurge.Muurger">
  <property name="muurgeRules">
    <list>
      <bean parent="majorityVote">
        <property name="properties">
          <map>
            <entry key="key" value="mcl:speciesName"/>
          </map>
        </property>
```

[11]net.straininfo2.semastri.integrators.EnvoIntegrator

[12]See http://environmentontology.org/

[13]be.ugent.twing.muurge.conditions.KeyExists

```
      </bean>
      <bean parent="enclosingRange">
        <property name="properties">
          <map>
            <entry key="key" value="mcl:optimalGrowthTemperature"/>
          </map>
        </property>
      </bean>
      <bean parent="aggregateToSet">
        <property name="properties">
          <map>
            <entry key="key" value="mcl:inhibitedBy"/>
          </map>
        </property>
      </bean>
    </list>
  </property>
</bean>
```

8.2 Discussion

The Muurge framework is intended as a practical and easy tool for semantic integration over a wide range of objects, beyond the applications in StrainInfo which served as the case study for its design. It is a practical solution to a practical problem and therefore many of the design decisions were made from a pragmatic perspective. Muurge is related to the work of A. Hunter and R. Summerton, who use formal logic-based techniques to merge structured reports with so-called fusion rules [80–83]. Their notion of structured report is similar to our concept of `Resource`. A fusion rule is a formula of meta-level first-order logic, and is written in a custom language. However, the theoretical aspects of the suggested logic might daunt the broad audience aimed for by the Muurge framework, and therefore we avoid the use of any mathematical language in the framework. In addition, Muurge is designed from the ground up to be incorporated as a library in other applications.

From an academic perspective it would be interesting to add an algorithm to automatically populate `Muurgers` with rules based on input. This can be done using both supervised and unsupervised learning approaches. The unsupervised learning approach would look at the input resources and make an educated guess about a suitable Muurge rule based on features in the input entries such as the data type, the variability of the values, whether they are functional, etc. A supervised learning algorithm can make use of example outputs to find Muurge rules that give similar outputs. These algorithms would work well on `Resources` extracted from the Semantic Web. Based on statements such as `owl:sameAs` `Resources` could be automatically extracted and the behaviour of properties could be automatically determined. In addition, there is also meta-information about those properties that might be helpful (e.g. whether they are functional).

In practice, resources are often related with each other in some way (e.g. a plain copy, resource based on other resource, inspired by,...). Therefore, the implicitly assumed independence of the sources might give a false feel of confidence. It would therefore be interesting to keep track of dependencies between resources. This information could dramatically change inconsistency detection. If an inconsistency originates at one source but has become widely spread in dependent sources, the more prevalent value might not necessarily be correct.

The idea of creating a framework for semantic integration was conceived after conversations with Prof. Eduard Hovy, Ph.D. from the USC Information Sciences Institute in Marina del Rey, Los Angeles, CA, USA (September 2010–July 2011).

9 Future perspectives

In what follows, we will discuss some possibly interesting future directions for the research presented in this dissertation. As the directions have been grouped by theme, this chapter is structured in sections about the StrainInfo platform, Histri, MCL and Semastri. Most of these future directions were already kept in mind when designing the StrainInfo platform architecture, and therefore the si2 codebase already contains some precursors to the features discussed below.

9.1 StrainInfo platform

Development of StrainInfo will never be finished, and there will never be a final release. As touched in Section 3.3, there is a continuous need for maintenance and support tasks to keep StrainInfo operational. Nevertheless, there is still a tremendous potential of unexplored research, which can form the basis of new features of the web application.

StrainInfo wants to continue to reach out to the microbiological community to illustrate the possibilities and the potential of widespread and freely available microbial information in an electronic format. The introduction and use of electronic microbiological information is relatively new, and its importance needs to be shown to microbiologists who possess data and need to be motivated to make it available in suitable formats. StrainInfo wants to do this by continuing to build applications that are able to generate output that otherwise would take a long time to create manually. An example of this is the quality assessment of gene sequences, i.e. what is the best 16S rRNA gene sequence available for a particular strain? This task is often done manually, but becoming harder due to the explosive growth of the number of available sequences. StrainInfo is in a position where it is able to select a *best* match from possibly multiple alternatives. By using all the available information, StrainInfo attempts to provide users a best match without manual intervention. These results can be used to automatically build a distance matrix and phylogenetic tree of a complete genus. This output is the result of composing several of our web services into a workflow. Additionally, a simplified web interface will be made available for users that are unable to run the workflow in a workflow management system.

Next, we aim at further including more semantic information into StrainInfo, which means that new information from new data sources will be incorporated. The necessary semantic integration is part of Semastri, and some aspects thereof have been discussed in Section 9.4. However, the resulting information will be presented on strain, taxon and other passports. As a consequence, StrainInfo will evolve towards a broad broker of microbiological information. This will attract a broader audience, and will translate to more page views. However, evolving towards more general-purpose passports has the downside that StrainInfo as a product may becomes less suited for some groups of specialized users.

Personalization will allow users to adapt their StrainInfo experience to their personal needs. This project is named MyStrainInfo. Depending on their background, users are interested in more detailed or other views of the information. For example, curators are interested in annotation differences between BRCs. Advanced users can be interested in the provenance of a passport page, i.e. the underlying resources and intelligence used to compile the page. In addition, MyStrainInfo will allow users to manage their own research collection. Users create new isolates or add cultures as subcultured from original BRC cultures. All user cultures also are assigned a culture URI, which solves the problem of ambiguous researchers' numbers. In addition, users will be able to attach information such as publications, fingerprinting results or other scientific results to their personal cultures. This will be done using MCL terms and this information will also be incorporated by Semastri. Lists of strains can be created, and these can be a starting point for further, automated analysis. All information will be shareable with co-operating users or the rest of the world.

Finally, culture URIs can point to different strain passports at different time points and this raises concerns about the repeatability of *in silico* experiments performed on information from StrainInfo. Even small changes to strain or other passports might have a considerable influence on the results of experiments. From a theoretical perspective, this can be avoided by versioning passports, and always using a reference to the specific version used. This reference can be as simple as a timestamp. However, it would make electronic information processing considerably more complex, and therefore it is not supported for the time being. In addition, strain passports already offer some versioning through the Histri logs. Nevertheless, the platform architecture has been foreseen to support versioning in the future if this would be required by complex applications. Versioning can be performed by assigning an integer StrainInfo revision number ('`si2_revision`') to every modification to the StrainInfo index. There are two important categories of modifications to the index: updating existing records (e.g. `Cultures` moved to another strain or linked to another predecessor in their Histri) and adding new records (e.g. synchronisation of new sequences or publications). To version updates of existing records, it is necessary to store the old versions of the record. For example, on every change to the `si2.cultures` table, the record can be copied to a history table. Based on the history table, it is possible to reconstruct the strain of a culture at a given StrainInfo revision or time point. To do this efficiently and elegantly, extra information (the previous strainId and Histri parent) must be stored on the history copy. For tables that extend only (i.e. addition of new records), it is sufficient to add the StrainInfo revision number (which acts as a timestamp) associated with the addition to the record. An interesting benefit of using a StrainInfo revision number is that this identifier also can be used as an anchor for provenance information. This fits in a generic system tracking all provenance information in StrainInfo, stored in a specialized set of tables. The possibilities of the information that can be kept track of is almost endless, but priority must be given to tracking changes belonging to synchronisation operations. Semastri is also compatible with a StrainInfo revision identifier, as well as the provenance information tracking subsystem.

9.2 Histri

Currently, Histris are constructed based on historical information. As this places the burden on the actual availability of such information, we need to evolve to a system where Histris are proactively maintained. How this wil be implemented partly depends on guidelines from BRC federations and networks such as the World Federation of Culture Collections (WFCC). Preferably, we need to evolve to a system similar to the registration system used by seed banks, using real-time registration of transfers at the time they are initiated (see Section 4.2.2). This can be done using a designated web service, accessible online through a web browser or as a built-in feature of BRC management software. A less radical solution can be the standardization of the textual history description format, combined with a required disclosure of at least the immediate predecessor. This guarantees that the culture can be linked to its ancestor when the BRC is synchronized with StrainInfo, eliminating the need for complex autobuild heuristics. As a result, the benefits of validated authenticity can be automatically obtained.

Histris can easily be expanded to be more fine-grained by, for example, adding internal BRC batch or lot numbers. As discussed in Section 4.2.4, adding more intermediate cultures decreases the information density of Histris. To keep Histris clear, the facility of grouping cultures needs to be introduced. All cultures (i.e. all batches or lots) held at a certain BRC can then be grouped. These groups can be collapsed to form a virtual node associated with the BRC, keeping the global overview clear. It can also be relevant to maintain information regarding the viability of the material and to hide possibly less relevant cultures. Next to the relatively simple technological additions, this depends on the willingness of BRCs to disclose internal information regarding their batches and viability of material.

Cultures added by MyStrainInfo users are preferably linked to the Histri. This strengthens the established authenticity of the users' cultures, which is important if user-added information is shared. Therefore, when adding a non-isolate culture to a personal collection in MyStrainInfo, connecting the culture to its Histri must be a central step in registering the culture. New isolates can not be connected to their Histri, but being responsible for an isolate might be an extra incentive to actively curate the Histri of the resulting strain. Having personal cultures linked to the global Histri of the strain makes it possible to make directed notifications to users when there are possible authenticity problems with their cultures. Owners of isolates can be informed about the proliferation of *their* strain, e.g. through automatic notifications of transfers to other BRCs, mentions in publications, etc.

However, grouping and hiding less relevant cultures is a prerequisite to keep the Histris uncluttered. In addition, there is also the necessity for privacy settings, allowing users to change the visibility of their cultures.

9.3 MCL

MCL is envisioned as an open, fundamental infrastructure for the microbiological community of the future. Next to the accessibility of biological material, a cornerstone of the Microbial Commons is the free distribution of the corresponding meta-information. Although StrainInfo used its unique position to create a language to fulfill its own practical needs, this is an opportunity to formalize the technological basis of universal electronic microbial information exchange. However, MCL is only able to grow beyond its initial scope through wide adoption by the community. Therefore, the microbiological community is encouraged to adopt the standard by creating and consuming data in this format and to provide feedback on the language structure, on existing or missing terms or on governance issues. Standards become truly valuable when there is a community consensus on their structure and a vision of universal usage materialized in the practical adoption of correctly formatted documents. This, combined with new developments such as the Semastri project can form the basis of a new era in microbiology where new insights might be gained from the vast amount of scientific knowledge readily available.

It is expected that an increasing number of BRCs will export their catalog in MCL format. StrainInfo will use the standard in a growing number of web services and exports. The standard will continue to evolve as a consequence of community feedback and the practical experience of an increasing number of MCL documents and tools becoming available. Extensions that include mappings to other data standards and addition of specialized terms (if necessary) will be carried out in modular case studies.

Many measurements and observations must be interpreted in their experimental context. Even trivial factors such as the growth temperature or the culture medium can have a large influence on the resulting colonies and their phenotypical properties. Therefore it is important to keep track of the biochemical context in which observations were made. Each MCL statement derived from experimental results therefore could be accompanied by its biochemical context, which is part of its provenance. This requires the MCL notion of experimental context, which could be expressed by a novel set of terms. Unfortunately, the experimental context is something that is hard to recover for information from readily existing sources, but a forced description of the experimental context might be an added value of MyStrainInfo. However, this signifies a practical change of how microbiological information is handled, viewed and interpreted, and the added complexity of the information might actually prove to be a large hindrance to a broad proliferation of MCL.

9.4 Semastri

The semantic integration result of Semastri improves as the amount of semantic information increases. The more information enters the system, the more coverage and confidence is created using the semantic integration algorithm. Therefore, the main challenge for Semastri is to find new sources of semantic information. The focus must be put on finding large-scale sources, as due to the broad range of strains in StrainInfo the effort only becomes visible if it covers a considerable number of strains. Furthermore, the resulting information can only be valued by new visualisations and flexible analysis tools.

The information extraction experiment we performed has many aspects that can be further improved. Evidently, the corpus itself can be expanded to include other parts of the publications in the current corpus, and other publications can be added as well. Detaching from the specific structure of species description sections requires significant modifications to the information extraction algorithm. In addition, the output of an extended algorithm can be curated in a manner similar to the Make Histri project: the algorithm is used to generate semantic information, which is verified by users afterwards. This way, the error margin of the algorithm can be increased to accommodate a larger recall. Nonetheless, the fundamental solution lies in an groundbreakingly different approach, which is already discussed in Section 7.2. Instead of using complex techniques to recover information captured in written text, the semantic information could be captured using relatively straightforward technology. However, this induces a paradigm shift were the publication is seen as a derived product of the semantic information produced for it, and the merit of producing semantic information must be credited as well.

MyStrainInfo has the potential to bring a large amount of user-generated fine-grained information to StrainInfo. However, this also poses new challenges, more specifically concerning privacy, personalized on-demand semantic integration, data quality and data granularity.

There is an important privacy aspect which needs to be looked into before many users will be willing to share information. A lot of information is kept confidential at its generation, to give researchers and their team the chance to verify and to further work with the information while having the advantage of being the first to have access to it. However, after publication, this information may be made available to a broader public, as it supports the publication and even might give additional visibility to the publication. Therefore, there must be enough privacy controls, based on notions of a social network (e.g. concepts such as team, collaborator, contact, everybody). The semantic integration algorithm needs to be aware of those privacy settings as it can only integrate information that is visible to the user.

Data quality is an issue when allowing users to add information, especially when that information is publicly visible. In an extreme case, the semantic integration could be fooled or overwhelmed by incorrect information in an attempt to fraud the system (e.g. create cultures that match popular search queries). However, information can be checked during input, the only moment where active curation can be guaranteed. If a strain is known to belong to a species, the semantic integration result of the species gives an idea about the range of values certain terms exhibit. If users try to enter information that falls outside those ranges or possibly even is inconsistent with the pre-existing information, a warning can be shown to the user to review the information. The quality of data originating from MyStrainInfo also depends on the popularity and adoption of the system. The more users independently input information about the same strain, the more reliable the semantic integration becomes. However, the sparsity issue due to the large number of strains has an impact which is hard to predict. If at any time, there is only a relatively small working set of *popular* or *trending* strains in present research, there is a larger chance on collisions of users independently entering information, and therefore an increased confidence in the presented information. Note that this working set is different from the working set of strains trending as the consequence of publication.

Next to the data quality itself, the semantic term coverage can also be improved. Although MCL term coverage has been expanding, the specificity of research demands personally tailored terms. These terms can be created by users in a Web 2.0 fashion part of MyStrainInfo. In addition, they are preferably reused by other users, so that data becomes interconnected on a larger scale, and does not tend to form data islands. There is a considerable community aspect involved in this: including community definition and curation of terms, community standard operating procedures, recommending potentially useful or related terms to stimulate term reuse, etc.

There are also creative unexpected large-scale sources of semantic information. A good example is fingerprints, such as the results of the well-known API® bacterial identification test strips produced by bioMérieux[1]. These profiles are often determined at a large scale, as they are performed as part of standard operation procedures. Each API profile is the combined result of a set of 10, 20 or 32 miniaturised experiments, encoded in a numeric representation. This numeric representation is used for identification, and is often stored in databases for later reference. However, it can also be decoded to a set of statements corresponding to the underlying phenotypical experiments performed in the API gallery. This is potentially interesting as those results are often available at a large scale in for example internal BRC databases.

The benefits of having semantic information electronically available at a large scale in a well-structured way remains largely unexplored and the most revolutionary applications might even be unimaginable at this moment. The logical step up is to combine the information in external ontologies with semantic microbial information. For example, the geographical location of the sample can be coupled to climate information. Aspects of the climate can then be correlated with phenotypical properties of cultures found in soil. Currently, references to only three external ontologies are made (GeoNames, EnvO and ChemSpider[2] [84] URIs) and therefore the bonding with the Linked Open Data can be improved by using more ontologies as MCL recommended vocabularies. The possibilities of correlating microbiological properties to new unexpected aspects are only limited by the imagination of microbiological researchers. From a computer science perspective, the technical hurdles of finding an answer to their queries will gradually become clear. The future probably will bring more advanced semantic information representation technologies and better algorithms to cope with information, effectively generating new knowledge, possibly even in a fully automatic way. However, its preparation starts now by fundamentally changing the way scientific information is collected, presented and shared.

[1]http://www.biomerieux.com
[2]http://www.chemspider.com

A Microbiological Common Language reference

This appendix contains definitions for the Microbiological Common Language (MCL) terms, as defined by the online MCL reference[1]. The online MCL reference is the authorative document for term definitions, and contains the most recent version. The term definitions are included here for the sake of completeness. The MCL reference does not define how MCL files should be formatted or used. More information and example MCL files can be found on the MCL project homepage[2].

MCL was heavily influenced by the MINE and CABRI projects. Therefore, the term definitions include information about mappings to other standards such as MINE, Dublin Core and PRISM. Depending on the term, this can be an exact mapping or requiring a reinterpretation of an existing field (marked as 'related'). In addition, to allow for an easier mapping from these standards, deprecated 'legacy' terms were added to the language. It is recommended to avoid the use of these deprecated terms in new applications. Therefore, definitions of deprecated terms

[1] `http://www.straininfo.net/projects/mcl/reference`

[2] `http://www.straininfo.net/projects/mcl/`

Prefix	Namespace
mcl:	`http://www.straininfo.net/ns/mcl/2.0/`
dc:	`http://purl.org/dc/elements/1.1/`
dcterms:	`http://purl.org/dc/terms/`
prism:	`http://prismstandard.org/namespaces/basic/2.0`
wgs84_pos:	`http://www.w3.org/2003/01/geo/wgs84_pos#`

Table A.1: Overview of namespaces used in MCL terms

contain information about possible alternatives.

For some terms, MCL strongly recommends the use of a predefined vocabulary. Terms with a controlled vocabulary list those recommended values in a table that is included with the term definition in the online MCL reference (but has been omitted in this appendix). This enables other users to interpret values in the intended way. The canonical value is the preferred value for persisting information using the term. In some cases, the canonical value is the same as used in the MINE standard, while in other cases, other canonical values were introduced. The alternative notations and comments columns help when mapping data from other data formats.

The value of some terms can only be interpreted given the experimental context and conditions in which they were determined. However, in practice, these conditions are often not available (e.g. on a BRC catalog entry or in a published species description), and implicitly assumed. As MCL was tailored to represent information from BRC catalogs and publications, some MCL terms implicitly or explicitly assume 'standard conditions'. MCL does not define what these are, but assumes that values for a culture or strain given by the same source are consistent. It remains the responsibility of the user to interpret the data with the necessary precaution. The `mcl:comments` term can be used to characterise the experimental conditions in a textual description.

All MCL terms are defined in the default MCL namespace, which is by convention abbreviated by `mcl:`. Terms from other namespaces are also used in MCL. An overview of all namespaces used by MCL can be found in Table A.1.

Entities

mcl:Culture – An instance of a strain, held at a given place and time.

Description:	According to the definition of Staley and Krieg, a bacterial culture is seen as a population of bacterial cells of a strain, instantiated in a given place during a given time, e.g. in a test tube, on an agar plate or in a cryopreserved or lyophilized state intended for long preservation. In practice, a mcl:Culture is associated with a strain number. Each issue of a new strain number yields a new culture. Reference: J.T. Staley and N.R. Krieg. Classification of procaryotic organisms: Bergeys Manual of Systematic Bacteriology, pages 1-4, 1984.

mcl:Strain – The result of the StrainInfo integration of cultures.

Description:	Staley and Krieg define a strain as the descendants of a single isolation in pure culture, usually a succession of cultures ultimately derived from a single colony. As a result, strain is associated with the integration results of StrainInfo. Reference: J.T. Staley and N.R. Krieg. Classification of procaryotic organisms: Bergeys Manual of Systematic Bacteriology, pages 1-4, 1984.

mcl:Sample – Sample from which a microorganism was isolated.

Description:	The environmental sample from which a microorganism was isolated. Multiple cultures can be isolated from the same Sample.

mcl:Isolation – The process of isolating a pure culture from an environmental sample.

Description:	The process of isolating a pure culture from an environmental sample. If multiple cultures have been isolated from a sample, each isolate corresponds to one Isolation entity.

mcl:Medium – A culture medium.

Description:	Culture medium used to grow cultures.

mcl:Publication – A scientific publication.

Description:	Describes a scientific publication. The term dcterms:bibliographicCitation is obligatory when describing publications. It contains a human-readable citation of the work, containing enough information to enable the user to find the intended publication. If the components of the citation are available separately (or if the citation can be easily split into components), it is recommended to include the separate components. The dc:creator field must be repeated for each individual author.

mcl:Deposit – The transition of one culture to another.

Description:	When a BRC transfers a culture to another BRC, a new culture originates: this process is characterized by the assignment of a new strain number. The process of deposit and assignment of new strain numbers results in a strain with an exchange history. StrainInfo visualises this exchange history as the Histri of the strain. The mcl:Deposit entity is used to model one transfer, along with meta-information, of the exchange history.

mcl:CatalogDescription – Metadata of a BRC catalog.

Description:	Metadata of a BRC catalog exported using the MCL 'catalog' XML schema.

mcl:BRC – Description of a BRC.

Description:	Used as part of the mcl:CatalogDescription element to describe a BRC (in the 'catalog' XML schema).

mcl:StrainInfo – Root element used in XML files.

Description:	This element is used by StrainInfo in XML files, as the root element of for example exports.

Terms

dc:creator – Author.

Description:	Author of article or BRC catalog MCL XML export.
Domain:	`mcl:Publication`, `mcl:CatalogDescription`
Range:	None (literal value)
Other standards:	Part of the DCMI Metadata Terms, see DCMI Metadata Terms, term dc:creator[3] for more information.

dc:title – Article title.

Description:	Title of publication.
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the DCMI Metadata Terms, see DCMI Metadata Terms, term dc:title[4] for more information.

dcterms:bibliographicCitation – Full bibliographic citation of publication.

Description:	Each publication must be described using a bibliographic citation. The citation must contain enough information to fully identify the publication. However, the format of the citation is not defined. If components such as publication title, authors or journal are available separately, these are preferably also listed separately using the terms dc:title, dc:creator, prism:publicationName, prism:volume, prism:number, prism:startingPage, prism:pageRange and dcterms:issued
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the DCMI Metadata Terms, see DCMI Metadata Terms, term dcterms:bibliographicCitation[5] for more information.

dcterms:created – Creation timestamp.

Description:	Creation date and time.
Domain:	`mcl:BRC`
Range:	None (literal value)
Other standards:	Part of the DCMI Metadata Terms, see DCMI Metadata Terms, term dc:created[6] for more information.

dcterms:issued – Year of issue

Description:	Date (year) when publication was issued.
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the DCMI Metadata Terms, see DCMI Metadata Terms, term dcterms:issued[7] for more information.

mcl:alkalinePhosphataseReaction – Alkaline phosphatase reaction.

Description:	Results of an alkaline phosphatase reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:betaGalactosidaseReaction – Beta-galactosidase reaction.

Description:	Results of a beta-galactosidase reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:catalaseReaction – Catalase reaction.

Description:	Results of a catalase reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:catalogLastUpdateDate – Timestamp of the last change to the BRC catalog.

Description:	Date and time of the last change to the BRC catalog, formatted as a ISO 8601 xsd:dateTime[8]
Domain:	`mcl:BRC`
Range:	None (literal value)

mcl:catalogURL – URL of corresponding online catalog entry.

Description:	URL containing a direct link to the online BRC catalog entry. If this term is used by StrainInfo, it will contain the URL of the BRC catalog entry in the strain browser. If this term is used by BRCs, it may contain a direct link to the BRC catalog entry. If such link does not exist, a link to an indermediary page can be given (e.g. an overview of a species) instead. However, this is disadvantageous as this can decrease the number of users reaching the correct catalog entry. Make sure that the URL does not resolve to an error page.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)

mcl:catalogVersion – Version of BRC catalog.

Description:	Text field which can be used by BRCs that have an internal catalog version number, for example associated with printed catalog releases. There are no requirements on the form or the included information and can be omitted if no internal version number is used.
Domain:	`mcl:BRC`
Range:	None (literal value)

mcl:cellGcContent – GC content.

Description:	Guanine-cytosine (GC) content of the cell DNA/RNA, expressed as a percentage (without "%").
Unit:	mol%
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: GC
Since:	MCL 2.1

mcl:cellLength – Average length of the cell, measured in μm.

Description:	Average length of the cell, measured in micrometer under standard conditions. The value can be one number or a range, unit is omitted.
Unit:	μm
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	MINE: see mcl:cellShape
Since:	MCL 2.1

mcl:cellShape – Shape of cells.

Description:	Shape of cells as measured under standard conditions.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Bacterial MINE: CELL, first subfield
Since:	MCL 2.1

mcl:cellSize – Average size of a cell, measured in μm.

Description:	Average size of a cell, measured in micrometer under standard conditions. The value contains both the average width and length (possibly as a range), separated by 'x'. Unit is omitted. Examples: 0.3–0.6x0.8–3.5 0.4x4.9
Unit:	μm
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: CELL, third subfield
Since:	MCL 2.1
Deprecated:	Use mcl:cellLength and mcl:cellWidth if possible.

mcl:cellWidth – Average width of the cell, measured in μm.

Description:	Average width of the cell, measured in micrometer under standard conditions. The value can be one number or a range, unit is omitted. The mcl:cellWidth needs to be less (or equal) than the mcl:cellLength.
Unit:	μm
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	MINE: see mcl:cellShape
Since:	MCL 2.1

mcl:colonyShape – Colony shape.

Description:	Typical shape of colonies.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:comments – Various comments and notes.

Description:	To be used to attach various comments or notes.
Domain:	`mcl:Culture`, `mcl:Strain`, `mcl:Medium`, `mcl:Sample`, `mcl:Deposit`, `mcl:Isolation`
Range:	None (literal value)
Deprecated:	This term is intended as a 'last resort' if data does not fit other terms.

mcl:cultureLastUpdateDate – Timestamp of the last change to a culture.

Description:	Date and time of the last change to the BRC catalog entry or record of a culture, formatted as a ISO 8601 xsd:dateTime[9].
Domain:	`mcl:Culture`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: EDM

mcl:cultureURI – Culture URI.

Description:	Culture URI of a mcl:Culture. When using RDF, the culture URI is explicitly used to refer to mcl:Culture resources. This term can be used to explicitly include the culture URI in XML files.
Domain:	`mcl:Culture`
Range:	None (literal value)

mcl:cytochromeOxidaseReaction – Cytochrome oxidase reaction.

Description:	Results of a cytochrome oxidase reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:depositDate – Date of deposit.

Description:	Date at which culture was deposited into other BRC. If unknown, month and/or day can be omitted. As opposed to MINE, MCL does not discriminate between date of receipt and date of accession. If both dates are known, but significantly different from each other, the earliest date (date of receipt) is preferred.
Domain:	`mcl:Deposit`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: EDR and EDA

mcl:depositor – Person depositing culture to other BRC.

Description:	Full name of person responsible for transferring the culture to a or another BRC.
Domain:	`mcl:Deposit`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: DEP

mcl:depositorInstitute – Institute depositing culture to a BRC.

Description:	Institute transferring the culture to a BRC. This is applicable for both the first deposit and subsequent deposits between BRCs.
Domain:	`mcl:Deposit`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: DEP

mcl:environmentPublication – A publication on isolation environment.

Description:	References a publication in which the original sample environment is described.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Publication`
More general form:	`mcl:publication`
Other standards:	Related to Bacterial MINE: ENHISLIT

mcl:gramReaction – Gram reaction.

Description:	Results of a Gram staining reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Bacterial MINE: GRAM
Since:	MCL 2.1

mcl:growthPH – pH level yielding growth.

Description:	pH level at which growth can be observed. This field can be used when the given pH level is not neccessary or not known to be optimal, but certainly yields growth when used for incubation. Unit is omitted.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: PHR and PHC
Since:	MCL 2.1

mcl:growthTemperature – Growth temperature.

Description:	Temperature at which growth can be observed. This field can be used when the given temperature is not neccessary or not known to be optimal, but certainly yields growth when used for incubation. Unit is omitted.
Unit:	°C
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial and Fungal MINE: TEGR

mcl:hasCulture – References a particular culture of a strain.

Description:	Links cultures to a strain. This field is only to be used by StrainInfo as this corresponds to the equivalence integration, integrating cultures into strains.
Domain:	`mcl:Strain`
Range:	`mcl:Culture`

mcl:hasSample – Environmental sample from which the microorganism was isolated.

Description:	Links a mcl:Culture to the mcl:Sample from which it was isolated. It therefore skips the intermediate step mcl:Isolation, which makes handling legacy BRC catalogs easier. Multiple cultures can be isolated from the same sample.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Sample`
Deprecated:	Since MCL 2.1, it is recommended to use mcl:isolatedFromSample instead. This field is retained to ensure compatibility with MCL 2.0.

mcl:history – Textual description of exchange history.

Description:	Textual description of exchange history of a culture, as often found in BRC catalogs. There are no strict formatting rules for this field as this field accepts the legacy descriptions found in BRC catalogs. Possibly contains extra annotations regarding date of deposit and depositors; and generally consists of arrows pointing left ('<–'), dates, and strain numbers.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: HIS (Bacterial MINE uses delimiter '<', Fungal MINE uses delimiter '>' and reverse order)

mcl:historyPublication – A publication on the (exchange) history of the microorganism.

Description:	References a publication in which strain history is described.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Publication`
More general form:	`mcl:publication`
Other standards:	Related to Bacterial MINE: ENHISLIT

mcl:id – Culture id.

Description:	This term can be used to explicitly refer to the cultureId of a culture. Note that the cultureId is included as part of the culture URI (see mcl:cultureURI), which can be resolved and dereferenced. Therefore, the latter is the preferred identifier.
Domain:	`mcl:Culture`
Range:	None (literal value)

mcl:indoleReaction – Indole reaction.

Description:	Results of an indole reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:inhibitedBy – Growth inhibitor.

Description:	Chemical (antibiotic) known to inhibit growth. List only one antibiotic per term: use multiple statements to indicate multiple antibiotics. See also inverse term mcl:notInhibitedBy.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Related to Bacterial MINE: TOL
Since:	MCL 2.1

mcl:isolatedFromSample – Isolation process from sample.

Description:	Isolation process from sample. Multiple isolations can be performed on the same Sample, but these are modelled using different Isolation entities using separate mcl:isolatedFromSample statements.
Domain:	`mcl:Isolation`
Range:	`mcl:Sample`
Since:	MCL 2.1

mcl:isolationDate – Date of isolation.

Description:	Date of isolation. If unknown, month and/or day can be omitted.
Domain:	`mcl:Isolation`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: ISOL, third subfield (delimited by ',')

mcl:isolationMethod – Isolation method used.

Description:	Method used to isolate the environmental sample into a pure culture. This term uses the same recommended vocabulary as Bacterial MINE.
Domain:	`mcl:Isolation`
Range:	None (literal value), but has set of recommended values.
Other standards:	Bacterial MINE: ISOM

mcl:isolator – Person responsible for isolation.

Description:	Full name of person responsible for isolation from environmental sample (isolator).
Domain:	`mcl:Isolation`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: ISOL

mcl:isolatorInstitute – Institute of isolator.

Description:	Institute of isolator at time of isolation. There is no mapping from MINE.
Domain:	`mcl:Isolation`
Range:	None (literal value)

mcl:maximalGrowthTemperature – Maximal growth temperature.

Description:	Maximal temperature (degree Celsius, do not include unit) at which growth on culture medium can be observed.
Unit:	°C
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
More general form:	`mcl:growthTemperature`
Other standards:	Bacterial MINE: TEGR, third subfield (Bacterial MINE: delimited by ';', Fungal MINE: delimited by '/')

mcl:maximumGrowthNACL – Maximal NaCl concentration for growth.

Description:	Maximal NaCl concentration at which growth can be observed. Unit is omitted.
Unit:	%
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: SALR and SALC
Since:	MCL 2.1

mcl:maximumGrowthPH – Highest pH for growth.

Description:	Highest pH level at which growth can be observed. Unit is omitted.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial MINE: PHC, third subfield (delimited by ';')
Since:	MCL 2.1

mcl:mediumDescription – Full description of culture medium preparation.

Description:	Full description (including list of ingredients and procedure) of culture medium preparation.
Domain:	`mcl:Medium`
Range:	None (literal value)

mcl:mediumName – Common name of medium.

Description:	Common name of culture medium. Example: 'TSA', 'Columbia Agar'
Domain:	`mcl:Medium`
Range:	None (literal value)

mcl:mediumNumber – 'Strain number' of a culture medium.

Description:	Medium identifier consisting of BRC acronym, the word 'Medium' and the internal BRC medium number/identifier. Example: 'LMG Medium 1'
Domain:	`mcl:Medium`
Range:	None (literal value)

mcl:mediumURL – URL at which a description of the medium can be found.

Description:	URL of the description of the culture medium in the online catalog. In cases where there is no separate medium page available, and the medium is listed directly on the culture catalog entry, an URL template needs to be provided. The following fields can be used and will be replaced depending on the particular context (i.e. current culture): • %STRAIN_NUMBER: replaced by the complete strain number • %STRAIN_NUMBER_NUMBER: replaced by the number portion of the strain number
Domain:	`mcl:Medium`
Range:	None (literal value)

mcl:minimalGrowthTemperature – Minimal growth temperature.

Description:	Minimal temperature (degree Celsius, do not include unit) neccessary to observe growth on culture medium.
Unit:	°C
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
More general form:	`mcl:growthTemperature`
Other standards:	Bacterial and Fungal MINE: TEGR, first subfield (Bacterial MINE: delimited by ';', Fungal MINE: delimited by '/')

mcl:minimumGrowthNACL – Minimum NaCl concentration for growth.

Description:	Minimum NaCl concentration at which growth can be observed. Unit is omitted.
Unit:	%
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: SALR and SALC
Since:	MCL 2.1

mcl:minimumGrowthPH – Minimum pH for growth.

Description:	Minimum pH level at which growth can be observed. Unit is omitted.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial MINE: PHC, first subfield (delimited by ';')
Since:	MCL 2.1

mcl:motile – Cells are motile.

Description:	Indicates if cells are motile or non-motile. If cells are motile, the mechanism can be indicated with mcl:motileBy.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Bacterial MINE: MOT, first subfield (delimited by ';')
Since:	MCL 2.1

mcl:motileBy – Motility mechanism used.

Description:	Motility mechanism used. If organism is not motile, this term is omitted. Recommended values correspond to Bacterial MINE.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Bacterial MINE: MOT, second subfield (delimited by ';')

mcl:noGrowthNACL – NaCl concentration yielding no growth.

Description:	NaCl concentration at which no growth can be observed. Unit is omitted.
Unit:	%
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: SALR and SALC
Since:	MCL 2.1

mcl:noGrowthPH – pH level yielding no growth.

Description:	pH level at which no growth can be observed. Unit is omitted.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: PHR and PHC
Since:	MCL 2.1

mcl:noGrowthTemperature – Temperature at which no growth can be observed.

Description:	Temperature at which no growth can be observed. Unit is omitted.
Unit:	°C
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial and Fungal MINE: TEGR

mcl:nomenclaturalPublication – A publication on nomenclature.

Description:	References the publication in which the species name was introduced. This citation is also part of the mcl:qualifiedSpeciesName.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Publication`
More general form:	`mcl:publication`

mcl:notInhibitedBy – Chemical not inhibiting growth.

Description:	Chemical (antibiotic) known to not inhibit growth, i.e. for which a culture or strain is resistant. Inverse of mcl:inhibitedBy. List only one antibiotic per term: use multiple statements to indicate multiple antibiotics.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Related to Bacterial MINE: TOL
Since:	MCL 2.1

mcl:notUtilizes – Organic compound not utilized by cell metabolism.

Description:	Organic compound not utilized by cell metabolism. Inverse of mcl:utilizes: explicitly states that organic compound is not utilized by the cell metabolism. List only one organic compound per term: use multiple statements to indicate multiple organic compounds.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:optimalGrowthNACL – NaCl concentration for optimal growth.

Description:	NaCl concentration at which optimal growth can be observed. Unit is omitted.
Unit:	%
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: SALR and SALC
Since:	MCL 2.1

mcl:optimalGrowthPH – pH for optimal growth.

Description:	Optimal pH level at which optimal growth can be observed. Unit is omitted.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial MINE: PHC, second subfield (delimited by ';')
Since:	MCL 2.1

mcl:optimalGrowthTemperature – Optimal growth temperature.

Description:	Temperature (degree Celsius, do not include unit) at which optimal growth on culture medium can be observed.
Unit:	°C
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
More general form:	`mcl:growthTemperature`
Other standards:	Bacterial MINE: TEGR, second subfield (Bacterial MINE: delimited by ';', Fungal MINE: delimited by '/')

mcl:originatingCulture – Original culture, before being deposited into another BRC.

Description:	Culture that will be deposited in another BRC. This field can only be used after resolving mcl:originatingStrainNumber to a culture.
Domain:	`mcl:Deposit`
Range:	`mcl:Culture`

mcl:originatingStrainNumber – Original strain number of culture, before being deposited in other BRC.

Description:	Original strain number of culture, before it is deposited into another BRC. This corresponds to the ancestor of the culture in the Histri.
Domain:	`mcl:Deposit`
Range:	None (literal value)

mcl:otherStrainNumber – Other, equivalent strain number.

Description:	Individual equivalent strain numbers can be given using this term. If multiple equivalent strain numbers are known, the term must be used for each individual equivalent strain number. Acronyms must be included in the strain number as this field is taken as is, and no extra context is assumed when processing this field. Note that mcl:Strain does not accept this property as all strain numbers are regarded as mcl:strainNumber of the given strain.
Domain:	`mcl:Culture`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: see mcl:otherStrainNumbers

mcl:otherStrainNumbers – A list of other, equivalent strain numbers.

Description:	Multiple equivalent strain numbers can be listed using this term. Only ; and = are accepted as delimiters, but can not be mixed. Whitespace before and after the delimiter is discarded.
Domain:	`mcl:Culture`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: OCC
Deprecated:	Use mcl:otherStrainNumber if possible; this field is provided for compatibility with legacy BRC catalogs.

mcl:oxidaseReaction – Oxidase reaction.

Description:	Results of an oxidase reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:oxygenRelationship – Oxygen relationship of organism.

Description:	Oxygen requirements of organism, i.e. aerobicity. Uses the same values as Bacterial MINE.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Bacterial MINE: OXR

mcl:preservationPublication – A publication on the preservation of the microorganism.

Description:	References a publication in which preservation methods (applicable to the culture or strain) are described.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Publication`
More general form:	`mcl:publication`
Other standards:	Bacterial MINE: PRELIT

mcl:publication – General publiciation.

Description:	Used to refer to a publication. When more information is known about the type of publication (e.g. nomenclatural publication, taxonomic publication, etc.), use one of the more precise terms.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Publication`

mcl:qualifiedSpeciesName – The qualified species name (including authors).

Description:	Species name, containing authors. Species name contains genus and species epithet, and if applicable, the subspecies epithet. Formatting follows rules of Bacterial and Fungal MINE.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: SP. SPP is also included in this term.
Since:	MCL 2.1

mcl:recommendMedium – Medium recommended for cultivation.

Description:	Recommended medium for cultivation of a culture; medium on which the culture will yield growth.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Medium`

mcl:resultingCulture – New culture after deposit in BRC.

Description:	Strain number of the culture in the receiving BRC. The latter appoints the strain number as part of the accession. This field can only be used after resolving mcl:resultingStrainNumber to a culture.
Domain:	`mcl:Deposit`
Range:	`mcl:Culture`

mcl:resultingStrainNumber – Newly assigned strain number after culture is deposited in BRC.

Description:	Strain number of the culture in the receiving BRC. The latter appoints the strain number as part of the accession.
Domain:	`mcl:Deposit`
Range:	None (literal value)

mcl:sampleCollector – Name of sample collector.

Description:	Full name of sample collector: responsible for the physical act of retrieving the Sample from the environment.
Domain:	`mcl:Sample`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: COLL

mcl:sampleCollectorInstitute – Institute of sample collector.

Description:	Institute of sample collector at time of sample collection.
Domain:	`mcl:Sample`
Range:	None (literal value)

mcl:sampleCulture – Culture derived from other culture.

Description:	This is the resolved form of mcl:sampleCultureStrainNumber, explicitly pointing to the culture.
Domain:	`mcl:Sample`
Range:	`mcl:Culture`

mcl:sampleCultureStrainNumber – Culture derived from other culture.

Description:	This term can be used when the microorganism was derived from another microorganism. A change in the genome sequence as the result of deliberate methods or actions is necessary to be able to use this term. Historic relations between cultures (i.e. the Histri) are expressed using the mcl:Deposit entity.
Domain:	`mcl:Sample`
Range:	None (literal value)

mcl:sampleDate – Date of sampling.

Description:	Date of sampling. If unknown, month and/or day can be omitted.
Domain:	`mcl:Sample`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: COLL, third subfield (delimited by ',')

mcl:sampleDescription – Full textual description of sample.

Description:	Full textual description of habitat and geographical location where sample was taken; may contain dates, collector names and a description of the sampling process.
Domain:	`mcl:Sample`
Range:	None (literal value)

mcl:sampleHabitat – Habitat where sample was taken.

Description:	Description of habitat where sample was taken. This can be fully textual, but it is recommended to use terms from ontologies such as EnvO. Note that this term is related to, but different from, Bacterial MINE: HABI and Fungal MINE: HAB. Those terms are used on a higher (species) level and document a known generality.
Domain:	`mcl:Sample`
Range:	None (literal value)
More general form:	`mcl:sampleDescription`
Other standards:	Bacterial MINE: ISOFR; Fungal MINE: SSTR

mcl:sampleLocationCountry – Country where sample was taken.

Description:	Country where the sample was taken. This can be done textually or using a GeoNames URI (preferred). For practical reasons, states, oceans and continents are also valid for inclusion in this term.
Domain:	`mcl:Sample`
Range:	None (literal value)
More general form:	`mcl:sampleLocationDescription`
Other standards:	Bacterial and Fungal MINE: LOC, first subfield (delimited by ';')

mcl:sampleLocationDescription – Geographic location where sample was taken.

Description:	Full description of geographic location where sample was taken. This includes, but is not limited to, names of countries, administrative areas, places, mountains, rivers, oceans and seas. Textual names and GeoNames URIs (preferred) are valid values for this term.
Domain:	`mcl:Sample`
Range:	None (literal value)
More general form:	`mcl:sampleDescription`
Other standards:	Bacterial and Fungal MINE: LOC

mcl:sampleLocationPlace – Place where sample was taken.

Description:	Description of the place where the sample was taken. This is the geographical location inside the country. Therefore, the country must not be included. The value can be text or a GeoNames URI (preferred). When using the latter, mcl:sampleLocationCountry can be omitted as it is redundant.
Domain:	`mcl:Sample`
Range:	None (literal value)
More general form:	`mcl:sampleLocationDescription`
Other standards:	Bacterial and Fungal MINE: LOC, second subfield (delimited by ';')

mcl:significantFattyAcid – Major fatty acid in cells

Description:	Major fatty acid found in cells. List only one fatty acid per term: use multiple statements to indicate multiple major fatty acids.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:significantPolarLipid – Major polar lipid of cells

Description:	Major polar lipid found in cells. List only one polar lipid per term: use multiple statements to indicate multiple polar lipids.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:significantQuinone – Significant quinones.

Description:	A significant ubi, isoprenoid or mena quinone.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:speciesName – The species name.

Description:	The species name of a culture or strain. Species name contains genus and species epithet, and if applicable, the subspecies epithet. Author names are not included, but can be referred to using the mcl:nomenclaturalPublication property.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: SP (partial). SPP is also included in this term.

mcl:sporeForming – Cells are spore forming.

Description:	Indicates whether cells form spores.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Other standards:	Related to Bacterial MINE: RESTST

mcl:strainNumber – A strain number.

Description:	A strainNumber identifies a culture. As a result, this field is obligatory for each mcl:Culture. Describing equivalent strain numbers can be done using the mcl:otherStrainNumber term. Acronyms must be included in the strain number as this field is taken as is, and no extra context is assumed when processing this field. Note that mcl:Strain corresponds to the equivalence integration result of multiple cultures, and by consequence, this term occurs once for each culture in the strain.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Bacterial and Fungal MINE: STN, ACCN

mcl:taxonomicPublication – A publication on taxonomy.

Description:	References the publication in which the taxonomy of the strain (or the species it belongs to) is described.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	`mcl:Publication`
More general form:	`mcl:publication`
Other standards:	Bacterial MINE: TAXLIT

mcl:typeStrainOf – Name of taxon if type strain.

Description:	If a strain is a type strain (or if a culture belongs to a type strain), the corresponding taxonomic name can be described using the typeStrainOf property. If the strain is type strain of multiple taxa, these names must be listed using multiple mcl:typeStrainOf statements.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
Other standards:	Related to Bacterial MINE: STAT

mcl:typeStrainOfGenus – Genus name if type strain of genus.

Description:	If a strain is a type strain (or if a culture belongs to a type strain), the corresponding genus can be described using the mcl:typeStrainOfGenus property. If the strain is type strain of multiple genera, all names must be listed using multiple statements.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
More general form:	`mcl:typeStrainOf`
Other standards:	Related to Bacterial MINE: STAT

mcl:typeStrainOfSpecies – Species name if type strain.

Description:	If a strain is a type strain (or if a culture belongs to a type strain), the corresponding species can be described using the mcl:typeStrainOfSpecies property. If the strain is type strain of multiple species, these species are listed using multiple mcl:typeStrainOfSpecies statements.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value)
More general form:	`mcl:typeStrainOf`
Other standards:	Related to Bacterial MINE: STAT

mcl:ureaseReaction – Urease reaction.

Description:	Results of an urease reaction: positive, negative, variable or unknown.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

mcl:utilizes – Organic compound utilized by cell metabolism.

Description:	Organic compound utilized by cell metabolism. List only one organic compound per term: use multiple statements to indicate multiple organic compounds. See also inverse property mcl:notUtilizes.
Domain:	`mcl:Culture`, `mcl:Strain`
Range:	None (literal value), but has set of recommended values.
Since:	MCL 2.1

prism:number – Issue number.

Description:	Issue number of article.
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the Publishing Requirements for Industry Standard Metadata (PRISM) standard, see specifications[10] for more information.

prism:pageRange – Page range of publication.

Description:	Page range of publication. When last page number is not known, prism:startingPage can be used. Field must contain two numbers, delimited by '–'. Example: "450–455"
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the Publishing Requirements for Industry Standard Metadata (PRISM) standard, see specifications[11] for more information.

prism:publicationName – Name of journal or book.

Description:	Name of journal or book where article was published.
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the Publishing Requirements for Industry Standard Metadata (PRISM) standard, see specifications[12] for more information.

prism:startingPage – First page number of publication.

Description:	First page number of publication. Can be used when last page number is not known.
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the Publishing Requirements for Industry Standard Metadata (PRISM) standard, see specifications[13] for more information.

prism:volume – Volume number.

Description:	Volume number of article.
Domain:	`mcl:Publication`
Range:	None (literal value)
Other standards:	Part of the Publishing Requirements for Industry Standard Metadata (PRISM) standard, see specifications[14] for more information.

wgs84_pos:alt – Altitude at which sample was taken.

Description:	Altitude is measured in metric meters, and the unit is omitted. Depth (e.g. for samples taken in caves, mines or oceans) is considered a negative altitude (and is measured relative to the elevation).
Domain:	`mcl:Sample`
Range:	None (literal value)
Other standards:	Part of WGS84 Geo Positioning RDF vocabulary[15]. Related Bacterial and Fungal MINE: LOC, second subfield (delimited by ';')
Since:	MCL 2.1 (replaces mcl:sampleAlt)

wgs84_pos:lat – Latitude at which sample was taken.

Description:	The WGS84 latitude at which the sample was taken (decimal degrees).
Domain:	`mcl:Sample`
Range:	None (literal value)
Other standards:	Part of WGS84 Geo Positioning RDF vocabulary[16]
Since:	MCL 2.1 (replaces mcl:sampleLat)

wgs84_pos:long – Longitude at which sample was taken.

Description:	The WGS84 longitude at which the sample was taken (decimal degrees).
Domain:	`mcl:Sample`
Range:	None (literal value)
Other standards:	Part of WGS84 Geo Positioning RDF vocabulary[17]
Since:	MCL 2.1 (replaces mcl:sampleLong)

Samenvatting

Het uitgangspunt van dit proefschrift is de vraag hoe door gebruik te maken van gepaste informaticatechnieken vooruitgang in de microbiologie kan geboekt worden. In Hoofdstuk 1 wordt daarom gestart met een snelcursus microbiologie, vanuit het oogpunt van de informatica. Na bemonstering van de omgeving en isolatie van microorganismen kunnen deze in reincultuur gebracht worden. Eenmaal in reincultuur, kunnen microorganismen *in vitro* verder gekweekt worden voor wetenschappelijk onderzoek en spreken we van culturen. Essentieel is de binaire celdeling die prokaryoten (bacteriën en archaea) vertonen: dit proces resulteert in twee kopieën van de oorspronkelijke cel. Behalve een klein effect van genetische wijzigingen, bevatten de nakomelingen hetzelfde genetisch materiaal als hun ouderlijke cel en worden bijgevolg equivalent genoemd. Door middel van speciale technieken zoals vriesdroging, kunnen culturen ook op lange termijn bewaard worden. Verschillende internationaal gevestigde collecties (Eng. 'Biological Resource Centers' of BRC's) bewaren en verspreiden dit materiaal. BRC's kennen identifiers toe aan culturen, de zogenaamde 'stamnummers', om het mogelijk te maken aan deze culturen te refereren. Dit leidt tot het concept 'stam' (Eng. 'strain'): de verzameling van alle equivalente culturen, voorgesteld door hun stamnummers. Het probleem dat onstaat bij deze veelheid aan stamnummers is dat als een onderzoeker alle informatie over een bepaalde stam wil, hij eigenlijk alle equivalente stamnummers moet gebruiken bij het zoeken.

Dit probleem wordt opgelost door StrainInfo (zie Hoofdstuk 2). StrainInfo is een online platform die informatie van equivalente culturen aanbiedt op een geïntegreerde manier. De informatie wordt gebundeld op zogenaamde 'strain passports' die een geïntegreerd overzicht per stam aanbiedt. Een strain passport bevat de lijst van equivalente stamnummers, informatie over het species waartoe de stam behoort, de uitwisselingsgeschiedenis van de stam, een lijst van relevante genoomsequenties en publicaties. Het strain passport geeft ook toegang tot de 'strain browser', die rechtstreekse toegang geeft naar de catalogusrecords van de onderliggende BRC's.

StrainInfo voert twee integratiestappen uit. Eerst is er de equivalentie-integratie, die door het berekenen van de transitieve sluiting van paargewijze equivalenties tussen culturen bepaalt tot welke stammen ze behoren. Dit wordt gevolgd door een semantische integratie, die op stamniveau een consensuswaarde berekent voor informatie die gebonden is aan culturen. Dit is het resultaat dat wordt getoond op het strain passport. Om problemen met niet-unieke stamnummers op te lossen, kent StrainInfo culture URI's (Uniform Resource Identifier) toe aan culturen. StrainInfo biedt het integratieresultaat ook aan in machine-leesbare formaten met behulp van SOAP (Simple Object Access Protocol) en RESTful (Representational State Transfer) webservices. Culture URI's geven toegang tot het corresponderende strain passport wanneer ze worden gebruikt in een webbrowser, of, gebaseerd op een techniek genaamd 'Content Negotiation', tot elektronische formaten.

Hoofdstuk 3 gaat dieper in op de technische implementatie van het StrainInfo platform. De huidige versie van StrainInfo ('si2') is gebaseerd op een prototype dat door zijn organische groei niet meer aan moderne softwareontwikkelingsvereisten voldeed. Er wordt gebruik gemaakt van een architectuur met drie lagen: een data laag (die overeenkomt met de databank en de code om deze aan te spreken), een tussenlaag (waar de hogere logica geïmplementeerd is) en een presentatielaag (die de webapplicatie bevat). De code is verdeeld in verschillende modules en wordt gecompileerd door middel van Maven. De tussenlaag bevat onder andere functionaliteit voor het parsen en opmaken van stamnummers (gebaseerd op code uit het prototype) en een systeem om de URL's van catalogi te genereren. De webapplicatie is geïmplementeerd met behulp van het Apache Struts2 framework, een zogenaamd Model-View-Controller (MVC) web framework. Om in de strain browser bepaalde catalogi toegankelijk te maken, werd een systeem ontwikkeld dat formulieren automatisch kan posten of sessies kan openen.

Om de kwaliteit van de equivalentie-integratie te verbeteren, werden de uitwisselingsgeschiedenissen van stammen gereconstrueerd. Hoofstuk 4 bespreekt hoe deze geschiedenissen kunnen voorgesteld worden als een boom (die een 'Histri' genoemd wordt), en hoe ze kunnen gereconstrueerd worden aan de hand van informatie in BRC catalogi en andere bronnen. Er bestaan twee grote soorten van overdrachten: 'distributionele overdrachten' (overdrachten tussen twee BRC's die tot doel hebben het materiaal verder te verspreiden) en 'institutionele overdrachten' (wanneer een groot deel van de stammen al dan niet fysisch overgedragen wordt als gevolg van institutionele wijzigingen zoals een overname, verhuis of naamswijziging). Als case study werd in het 'Make Histri' project gepoogd de Histri van alle prokaryotische typestammen manueel te cureren. Het project wordt gecoördineerd vanuit een homepagina die een overzicht van alle typestammen en de vooruitgang in het curatiewerk bevat. De Histris worden gereconstrueerd in de Histri editor. Deze tool is uitgerust met een systeem dat een semi-automatische voorspelling van de gecureerde Histri genereert, die door de curator dient nagekeken en aangevuld te worden.

Hoofdstuk 5 bespreekt enkele technische aspecten van het 'Make Histri' project. De Histri editor is ontwikkeld als een alleenstaande Java Swing desktopapplicatie, en maakt gebruik van webservices om te communiceren met StrainInfo. Om de boom grafisch en met drag-and-drop editeerbaar te maken, werd een aangepast GUI paneel ontwikkeld dat bovendien geanimeerd is en over verschillende vormgevingen beschikt. Het systeem dat een semi-automatische voorspelling van de Histri maakt is gebaseerd op een onderliggende datastructuur die individuele suggesties (voorspellingen) modelleert. Suggesties worden gegenereerd door verschillende generatoren, waaronder een generator die domeinkennis van institutionele overdrachten bezit en een generator die ingebouwde tekstuele geschiedenisbeschrijvingen verwerkt. Deze datastructuur wordt gebruikt door het algoritme ('autobuild') dat de voorspelling genereert. Omdat de webservices enkel na authenticatie kunnen gebruikt worden (bescherming tegen vandalisme), wordt een truukje gebruikt om de authenticatie van de aangemelde gebruiker op de webapplicatie door te geven aan de Histri editor.

Na het verbeteren van de kwaliteit van de equivalentie-integratie, is de volgende vraag hoe méér informatie kan worden toegevoegd om het systeem rijker aan precieze informatie te maken. Deze informatie noemen we semantische informatie. We ontwikkelden de Microbiological Common Language (MCL), een datastandaard voor het elektronisch uitwisselen van microbiologische informatie, die is beschreven in Hoofstuk 6. Deze datastandaard werd sterk beïnvloed door de MINE (Microbial Information Network in Europe) standaarden die eind jaren tachtig verschenen. MCL definieert entiteiten (die overeenkomen met objecten) en termen (die overeenkomen met eigenschappen), maar legt geen representatietechnologie vast. MCL wordt gebruikt voor het synchronizeren van BRC's met StrainInfo ('catalog schema') en het verspreiden de integratieresultaten als elektronische informatie ('strain schema'). MCL bestanden volgens het catalog schema worden actief gebruikt door BRC's om te synchronizeren met StrainInfo. De volledige referentie van alle MCL entiteiten en termen wordt gegeven in Appendix A.

Semastri (uitgesproken met een knipoog naar 'chemistry') is het project dat overeenkomt met het verzamelen, semantisch integreren en voorstellen van gedetailleerde semantische informatie en wordt beschreven in Hoofdstuk 7. Het resultaat van Semastri wordt getoond in een nieuwe module op het strain en taxon passport dat MCL velden en hun overeenkomstige consensuswaarden bevat. De bedoeling van deze informatie is om StrainInfo voor een breder publiek relevant te maken.

Informatie komt StrainInfo binnen op cultuurniveau en wordt voorgesteld als 'culture resources', een sleutel-waarde-paren datastructuur. Culture resources worden geclusterd (equivalentieintegratie) om daarna samengevoegd te worden (semantische integratie). Het resultaat is een 'strain resource', die de consensuswaarden op stammenniveau voorstelt. Semastri gebruikt semantische informatie geleverd via de MCL XML synchronisatie, Make Histri project en oude informatie uit crawls van BRC catalogi. Daarnaast werd een experiment gestart om semantische informatie uit literatuur te extraheren. Als case study wordt semantische informatie geëxtraheerd uit species beschrijvingen in het tijdschrijft *International Journal of Systematic and Evioluationary Microbiology* (IJSEM). Een pijplijn werd geïmplementeerd volgens het Apache UIMA (Unstructured Information Management) framework. Deze pijplijn extraheert de semantische informatie aan de hand van een reeks annotator componenten die achtereenvolgend worden uitgevoerd. Elke component voegt annotaties aan de oorspronkelijke tekst toe, en deze annotaties kunnen door andere componenten gebruikt worden. Het resultaat van de pijplijn is een verzameling culture resources die de semantische informatie uitgedrukt met MCL termen bevatten.

Het semantische integratiealgoritme is een samenstelling van regels die een consensuswaarde genereren voor elke MCL term die voorkomt in de verzameling culture resources. Elke term wordt behandeld door een aangepaste heurstiek, zoals gelijkheid (alle waarden moeten dezelfde zijn), meerderheidsstemming, aggregatie tot een verzameling, berekening van het omsluitend interval en redeneren met ontologieën. Het algoritme is geïmplementeerd met behulp van het Muurge framework. Door zijn algemene inzetbaarheid werd dit framework losgekoppeld van StrainInfo. Het wordt besproken in Hoofdstuk 8. De basisdatastructuur is de `Resource`, die gezien kan worden als een veralgemening van de culture resource. Het effectieve integratiealgoritme wordt geïmplementeerd in een `Muurger`. `Muurgers` bestaan uit verschillende `MuurgeRules`, regels die individuele velden semantisch integreren. Het framework voorziet de concepten `Condition` en `Function` om maximale flexibiliteit en herbruikbaarheid van regels te garanderen. Het framework kan geconfigureerd worden met behulp van een Spring configuratiebestand (declaratieve configuratie), of rechtstreeks in code.

We eindigen de verhandeling met een overzicht van potentieel interessante verdere onderzoeksrichtingen voor StrainInfo (zie Hoofdstuk 9. Voor elke onderzoeksrichting wordt besproken hoe deze past in 'MyStrainInfo', de Web 2.0 visie van StrainInfo.

Scientific outreach

During the research that lead to this dissertation, Bert Verslyppe has contributed to the following publications. This manuscript contains work adapted from publications marked with ⋆.

- B. Van Brabant, T. Gray, B. Verslyppe, N. Kyrpides, K. Dietrich, F.O. Glöckner, J. Cole, R. Farris, L.M. Schriml, P. De Vos, B. De Baets, D. Field, and P. Dawyndt. Laying the foundation for a Genomic Rosetta Stone: creating information hubs through the use of consensus identifiers. *OMICS A Journal of Integrative Biology*, 12(2):123–127, 2008.
- B. Verslyppe, R. Kottmann, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. Microbiological Common Language (MCL): a standard for electronic information exchange in the Microbial Commons. *Research in Microbiology*, 161(6):439–445, 2010. ⋆
- B. Verslyppe, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. Make Histri: reconstructing the exchange history of bacterial and archaeal type strains. *Systematic and Applied Microbiology*, 34(5):328–336, 2011. ⋆
- N. Morrison, D. Hancock, L. Hirschman, P. Dawyndt, B. Verslyppe, N. Kyrpides, R. Kottmann, P. Yilmaz, F.O. Glöckner, J. Grethe, T. Booth, P. Sterk, G. Nenadic, D. Field. Data shopping in an open marketplace: Introducing the Ontogrator web application for marking up data using ontologies and browsing using facets. *Standards in Genomic Sciences*, 4(2):286–292, 2011.
- B. Verslyppe, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. StrainInfo introduces electronic passports for microorganisms. *PLoS ONE*, 2012. (submitted) ⋆

Bert Verslyppe has contributed to the following conference papers:

- B. Verslyppe, B. Slabbinck, W. De Smet, P. De Vos, B. De Baets, and P. Dawyndt. StrainInfo.net Web Services: Enabling Microbiologic Workflows Such as Phylogenetic Tree Building and Biomarker Comparison. *IEEE Fourth International Conference on eScience, 2008. eScience'08*, pages 603–607, 2008.

The following posters were presented at conferences or workshops:

- "Reconstructing the exchange history of bacterial and archaeal strains" (Bert Verslyppe, Paul De Vos, Bernard De Baets, Peter Dawyndt). Presented at the 27th annual meeting of the European Culture Collections' Organization (ECCO), 10-11/06/2008, NH Gent Belfort, Gent. Won third poster price.
- "A minimal and extensible standard for the exchange of microbial information" (Bert Verslyppe, Bernard De Baets, Paul De Vos, Peter Dawyndt). Presented at the 27th annual meeting of the European Culture Collections' Organization (ECCO), 10-11/06/2008, NH Gent Belfort, Gent.
- "StrainInfo: your microbial stepping stone" (Bert Verslyppe, Wim De Smet, Wim Gillis, Bernard De Baets, Paul De Vos, Peter Dawyndt). Presented at various workshops and conferences. Won first poster price on 12th International Conference on Culture Collections (ICCC12), Florianópolis, Santa Catarina, Brazil, 26/09/2010-01/10/2010, represented by Peter Dawyndt.
- "Make Histri: calling on the microbiological community to curate its historical data" (Bert Verslyppe, Wim De Smet, Bernard De Baets, Paul De Vos, Peter Dawyndt). Presented at the 7th Summer School on Ontological Engineering and the Semantic Web (SSSW'09), 5-11/07/2009, Residencia Lucas Olazabal, Cercedilla, Spain.
- "Microbiological Common Language (MCL): a standard for electronic information exchange in the Microbial Commons" (Bert Verslyppe, Wim De Smet, Bernard De Baets, Paul De Vos, Peter Dawyndt).
- "Semantic integration of isolation habitat and location in StrainInfo" (Bert Verslyppe, Wim De Smet, Paul De Vos, Bernard De Baets, Peter Dawyndt). Presented at the Workshop on Advances in Bio Text Mining (BioTM 2010), 10-11/05/2010, VIB, Ghent, Belgium and the 10th GSC (Genomics Standards

Consortium) Workshop (GSC10), 4-6/10/2010, Argonne National Laboratory, Argonne, IL, USA.

Bibliography

[1] W.B. Whitman, D.C. Coleman, and W.J. Wiebe. Prokaryotes: the unseen majority. *Proceedings of the National Academy of Sciences*, 95(12):6578–6583, 1998.

[2] M.G.A. Van Der Heijden, R.D. Bardgett, and N.M. Van Straalen. The unseen majority: soil microbes as drivers of plant diversity and productivity in terrestrial ecosystems. *Ecology Letters*, 11(3):296–310, 2008.

[3] M.T. Madigan, J.M. Martinko, D. Stahl, and D.P. Clark. Benjamin Cummings, 2010.

[4] C.R. Woese. Bacterial evolution. *Microbiological reviews*, 51(2):221–271, 1987.

[5] T. Berners-Lee. Information management: A proposal. Technical Report CERN-DD-89-001-OC, CERN, Geneva, Mar 1989.

[6] T. Berners-Lee and R. Cailliau. WorldWideWeb: Proposal for a HyperText project. 1990.

[7] B. Smith and A.C. Varzi. Surrounding Space The Ontology of Organism-Environment Relations. *Theory in Biosciences*, 121(2):139–162, 2002.

[8] R.G. Eagon. Pseudomonas natriegens, a marine bacterium with a generation time of less than 10 minutes. *Journal of Bacteriology*, 83(4):736–737, 1962.

[9] M.M. Mason. A comparison of the maximal growth rates of various bacteria under optimal conditions. *Journal of Bacteriology*, 29(2):103–110, 1935.

[10] J.T. Staley and N.R. Krieg. Classification of procaryotic organisms: an overview. *Bergeys Manual of Systematic Bacteriology*, pages 1–4, 1984.

[11] L. Dijkshoorn, B.M. Ursing, and J.B. Ursing. Strain, clone and species: comments on three basic concepts of bacteriology. *Journal of Medical Microbiology*, 49(5):397–401, 2000.

[12] S.P. Lapage, P.H.A. Sneath, E.F. Lessel, V.B.D. Skerman, H.P.R. Seeliger, and W.A. (editors) Clark. International Code of Nomenclature of Bacteria (1990 Revision). pages 17–20, 1992.

[13] D. Smith. Culture collections over the world. *International Microbiology*, 6(2):95–100, 2003.

[14] D. Emerson and W. Wilson. Giving microbial diversity a home. *Nature Reviews Microbiology*, 7(11):758, 2009.

[15] E. Stackebrandt. Diversification and focusing: strategies of microbial culture collections. *Trends in Microbiology*, 18:283–287, 2010.

[16] OECD. Biological Resource Centres. Underpinning the future of life sciences and biotechnology. 2001.

[17] OECD. OECD Best Practice Guidelines for Biological Resource Centres. 2007.

[18] R.S. Breed, E.G.D. Murray, and N.R. Smith. *Bergey's Manual of Determinative Bacteriology,*. Williams and Wilkins Co, Baltimore, MD, United States, 1957.

[19] P. Vandamme, B. Pot, M. Gillis, P. De Vos, K. Kersters, and J. Swings. Polyphasic taxonomy, a consensus approach to bacterial systematics. *Microbiological reviews*, 60(2):407–438, 1996.

[20] K.T. Konstantinidis, A. Ramette, and J.M. Tiedje. The bacterial species definition in the genomic era. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 361(1475):1929–1940, 2006.

[21] P. De Vos and H.G. Trüper. Judicial Commission of the International Committee on Systematic Bacteriology. IXth International (IUMS) Congress of Bacteriology and Applied Microbiology. Minutes of the meetings, 14, 15 and 18 August 1999, Sydney, Australia. *International Journal of Systematic and Evolutionary Microbiology*, 50(6):2239–2244, 2000.

[22] S.P. Lapage, P.H.A. Sneath, E.F. Lessel, et al. *International Code of Nomenclature of Bacteria: Bacteriological Code, 1990 Revision.* American Society for Microbiology, 1992.

[23] L.D. Stein. Integrating biological databases. *Nature Reviews Genetics*, 4(5):337–345, 2003.

[24] T. Coenye, E. Mahenthiralingam, D. Henry, J.J. LiPuma, S. Laevens, M. Gillis, D.P. Speert, and P. Vandamme. Burkholderia ambifaria sp. nov., a novel member of the burkholderia cepacia complex including biocontrol and cystic fibrosis-related isolates. *International journal of systematic and evolutionary microbiology*, 51(4):1481–1490, 2001.

[25] S. Brunak, A. Danchin, M. Hattori, H. Nakamura, K. Shinozaki, T. Matise, and D. Preuss. Nucleotide Sequence Database Policies. *Science*, 298(5597):1333, 2002.

[26] J.P. Euzéby. List of Bacterial Names with Standing in Nomenclature: a folder available on the internet. *International Journal of Systematic and Evolutionary Microbiology*, 47(2):590–592, 1997.

[27] P.W. Crous, W. Gams, J.A. Stalpers, V. Robert, and G. Stegehuis. MycoBank: an online initiative to launch mycology into the 21st century. *Studies in Mycology*, 50:19–22, 2004.

[28] S. Federhen. The NCBI taxonomy database. *Nucleic Acids Research*, 40(Database Issue):D136–D143, 2012.

[29] D.A. Benson, M.S. Boguski, D.J. Lipman, J. Ostell, B.F. Ouellette, B.A. Rapp, and D.L. Wheeler. GenBank. *Nucleic Acids Research*, 27(1):12, 1999.

[30] G. Stoesser, M.A. Tuli, R. Lopez, and P. Sterk. The EMBL nucleotide sequence database. *Nucleic Acids Research*, 27(1):18, 1999.

[31] Y. Tateno, S. Miyazaki, M. Ota, H. Sugawara, and T. Gojobori. DNA Data Bank of Japan (DDBJ) in collaboration with mass sequencing teams. *Nucleic Acids Research*, 28(1):24, 2000.

[32] E. Pruesse, C. Quast, K. Knittel, B.M. Fuchs, W. Ludwig, J. Peplies, and F.O. Glöckner. SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucleic Acids Research*, 35(21):7188–7196, 2007.

[33] D.L. Wheeler, T. Barrett, D.A. Benson, S.H. Bryant, K. Canese, V. Chetvernin, D.M. Church, M. DiCuccio, R. Edgar, S. Federhen, L.Y. Geer, Y. Kapustin,

O. Khovayko, D. Landsman, D.J. Lipman, T.L. Madden, D.R. Maglott, J. Ostell, V. Miller, K.D. Pruitt, G.D. Schuler, E. Sequeira, S.T. Sherry, K. Sirotkin, A. Souvorov, G. Starchenko, R.L. Tatusov, T.A. Tatusova, L. Wagner, and E. Yaschenko. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 35(Database issue):D5–D12, 2007.

[34] L.D. Stein. Creating a bioinformatics nation. *Nature*, 417(6885):119–120, 2002.

[35] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[36] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):W729–W732, 2006.

[37] P. Dawyndt, M. Vancanneyt, H. De Meyer, and J. Swings. Knowledge accumulation and resolution of data inconsistencies during the integration of microbial information sources. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1111–1126, 2005.

[38] B. Verslyppe, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. Make Histri: reconstructing the exchange history of bacterial and archaeal type strains. *Systematic and Applied Microbiology*, 34(5):328–336, 2011.

[39] J. Ma, S. Miyazaki, and H. Sugawara. A handy database for culture collections worldwide: CCINFO-PC. *Bioinformatics*, 11(2):209–212, 1995.

[40] J. Stalpers, M. Kracht, D. Janssens, J. Deley, J. Vandertoorn, J. Smith, D. Claus, and H. Hippe. Structuring strain data for storage and retrieval of information on bacteria in MINE, the Microbial Information Network Europe. *Systematic and Applied Microbiology*, (13):92–103, 1990.

[41] W. Gams, G.L. Hennebert, J.A. Stalpers, D. Janssens, M.A. Schipper, J. Smith, D. Yarrow, and D.L. Hawksworth. Structuring strain data for storage and retrieval of information on fungi and yeasts in MINE, the Microbial Information Network Europe. *Journal of General Microbiology*, 134(6):1667–89, 1988.

[42] D. Field, L. Amaral-Zettler, G. Cochrane, J.R. Cole, P. Dawyndt, G.M. Garrity, J. Gilbert, F.O. Glöckner, L. Hirschman, I. Karsch-Mizrachi, H.P. Klenk, R. Knight, R. Kottmann, N. Kyrpides, F. Meyer, I. San Gil, S.A. Sansone, L.M. Schriml, P. Sterk, T. Tatusova11, D.W. Ussery, O. White, and J. Wooley. The genomic standards consortium. *PLoS Biology*, 9(6):e1001088, 2011.

[43] B. Verslyppe, R. Kottmann, W. De Smet, B. De Baets, P. De Vos, and P. Dawyndt. Microbiological Common Language (MCL): a standard for electronic information exchange in the Microbial Commons. *Research in Microbiology*, 161(6):439–445, 2010.

[44] P. Romano, O. Aresu, M. Assunta Manniello, and B. Parodi. Interoperability of CABRI services and biochemical pathways databases. *Comparative and Functional Genomics*, 5(2):169–172, 2004.

[45] P. Romano, P. Dawyndt, F. Piersigilli, and J. Swings. Improving interoperability between microbial information and sequence databases. *BMC Bioinformatics*, 6(Suppl 4):S23, 2005.

[46] T. Nilsen, I.F. Nes, and H. Holo. Enterolysin A, a cell wall-degrading bacteriocin from Enterococcus faecalis LMG 2333. *Applied and Environmental Microbiology*, 69(5):2975–2984, 2003.

[47] G.M. Garrity, L.M. Thompson, D.W. Ussery, N. Paskin, D. Baker, P. Desmeth, D.E. Schindel, and P.S. Ong. Studies on Monitoring and Tracking Genetic Resources: an Executive Summary. *Standards in Genomic Sciences*, 1(1):78–86, 2009.

[48] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic Syntax. *RFC 3986*, 2005.

[49] T. Clark, S. Martin, and T. Liefeld. Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics*, 5(1):59–70, 2004.

[50] N. Paskin. Digital Object Identifiers. *Information Services and Use*, 22(2):97–112, 2002.

[51] L. Sauermann, R. Cyganiak, D. Ayers, and M. Völkel. Cool URIs for the Semantic Web. *W3C Interest Group*, 2008.

[52] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. *RFC 2616*, 1999.

[53] O. Lassila and R.R. Swick. Resource Description Framework (RDF): model and syntax. *W3C Recommendation*, 1999.

[54] F. Manola, E. Miller, and B. McBride. RDF primer. *W3C Recommendation*, 2004.

[55] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[56] T. Clark. Editorial: Identity and interoperability in bioinformatics. *Briefings in Bioinformatics*, 4(1):4–6, 2003.

[57] T. O'Brien, J. Casey, B. Fox, J. Van Zyl, M. Moser, E. Redmond, and L. Shatzer. *Maven: The Complete Reference*. O'Reilly Media, 2010.

[58] C. Walls. *Spring in Action, Third Edition*. Manning Publications Co., 2011.

[59] D. Brown, C.M. Davis, and S. Stanlick. *Struts 2 in Action*. Manning Publications Co., 2008.

[60] I. Roughley. *Starting Struts 2*. C4Media, 2007.

[61] FC Harrison. The discoloration of halibut. *Canadian Journal of Research*, 1(3):214–239, 1929.

[62] J. Brisou, C. Tysset, and B. Vacher. Study of 3 microbial strains of the family Pseudomonadaceae, whose synergism induces a septicemic-like disease in white fishes of the Dordogne & Lot Rivers & their tributaries. *Annales de l'Institut Pasteur*, 96:689–696, 1959.

[63] B.D Skerman, V. McGowan, and P.H.A. Sneath. Approved lists of bacterial names. *International Journal of Systematic Bacteriology*, 30:225–420, 1980.

[64] P. Vandamme, J.F. Bernardet, P. Segers, K. Kersters, and B. Holmes. New perspectives in the classification of the flavobacteria: Description of chryseobacterium gen. nov., bergeyella gen. nov., and empedobacter nom. rev. *International Journal of Systematic and Evolutionary Microbiology*, 44(4):827–831, 1994.

[65] Food and Agriculture Organization of the United Nations (FAO). International treaty on Plant Genetic Resources for Food and Agriculture. 2001.

[66] H. Cooper. The International Treaty on Plant Genetic Resources for Food and Agriculture. *Review of European Community and International Environmental Law*, 11(1):1–16, 2002.

[67] A. Willems, B. Hoste, J. Tang, D. Janssens, and M. Gillis. Differences between subcultures of the Mesorhizobium loti type strain from different culture collections. *Systematic and Applied Microbiology*, 24(4):549–553, 2001.

[68] L. Von Ahn and L. Dabbish. Labeling images with a computer game. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 319–326, 2004.

[69] L. Von Ahn and L. Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[70] C. Haase and R. Guy. *Filthy Rich Clients: developing animated and graphical effects for desktop Java applications*, chapter 14 and 15. Prentice Hall, 2008.

[71] P. Dawyndt, T. Dedeurwaerdere, and J. Swings. Exploring and exploiting Microbiological Commons: contributions of bioinformatics and intellectual property rights in sharing biological information. *International Social Science Journal*, 188:249–258, 2006.

[72] T. Dedeurwaerdere, P. Desmeth, P. Green, et al. Microbiological resources, who owns what? building the Microbial Commons. . *WFCC and DSMZ, Goslar*, pages 129–130, 2007.

[73] D. Field, G. Garrity, T. Gray, N. Morrison, J. Selengut, P. Sterk, T. Tatusova, N. Thomson, M.J. Allen, S.V. Angiuoli, M. Ashburner, N. Axelrod, S. Baldauf, S. Ballard, J. Boore, G. Cochrane, J. Cole, P. Dawyndt, P. De Vos, C. de Pamphilis, R. Edwards, N. Faruque, J. Feldman, R. Gilbert, P. Gilna, F.O. Glöckner, P. Goldstein, R. Guralnick, D. Haft, D. Hancock, H. Hermjakob, C. Hertz-Fowler, P. Hugenholtz, I. Joint, L. Kagan, M. Kane, J. Kennedy, G. Kowalchuk, R. Kottmann, E. Kolker, S. Kravitz, N. Kyrpides, J. Leebens-Mack, S.E. Lewis, K. Li, A.L. Lister, P. Lord, N. Maltsev, V. Markowitz, J. Martiny, B. Methe, I Mizrachi, R Moxon, K. Nelson, J. Parkhill, L. Proctor, O. White, S.A. Sansone, A. Spiers, R. Stevens, P. Swift, C. Taylor, Y. Tateno,

A. Tett, S. Turner, D. Ussery, B. Vaughan, N. Ward, T. Whetzel, I. San Gil, G. Wilson, and A. Wipat. The Minimum Information about a Genome Sequence (MIGS) specification. *Nature Biotechnology*, 26(5):541–547, 2008.

[74] R. Kottmann, T. Gray, S. Murphy, L. Kagan, S. Kravitz, T. Lombardot, D. Field, and F.O. Glockner. A standard MIGS/MIMS compliant XML schema: Toward the Development of the Genomic Contextual Data Markup Language (GCDML). *OMICS A Journal of Integrative Biology*, 12(2):115–121, 2008.

[75] B. Van Brabant, T. Gray, B. Verslyppe, N. Kyrpides, K. Dietrich, F.O. Glöckner, J. Cole, R. Farris, L.M. Schriml, P. De Vos, B. De Baets, D. Field, and P. Dawyndt. Laying the foundation for a Genomic Rosetta Stone: creating information hubs through the use of consensus identifiers. *OMICS A Journal of Integrative Biology*, 12(2):123–127, 2008.

[76] R. Sætre, K. Yoshida, A. Yakushiji, Y. Miyao, Y. Matsubyashi, and T. Ohta. AKANE system: protein-protein interaction pairs in BioCreAtIvE2 challenge, PPI-IPS subtask. pages 209–212, 2007.

[77] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American Magazine*, 284(5):34–43, 2001.

[78] T. Finin, L. Ding, L. Zhou, and A. Joshi. Social networking on the semantic web. *The Learning Organization*, 12(5):418–435, 2005.

[79] L. Hirschman, C. Clark, K.B. Cohen, S. Mardis, J. Luciano, R. Kottmann, J. Cole, V. Markowitz, N. Kyrpides, N. Morrison, L.M. Schriml, D. Field, and the Novo Project. Habitat-lite: a gsc case study based on free text terms for environmental metadata. *OMICS A Journal of Integrative Biology*, 12(2):129–136, 2008.

[80] A. Hunter. Merging potentially inconsistent items of structured text. *Data & Knowledge Engineering*, 34(3):305–332, 2000.

[81] A. Hunter and R. Summerton. Fusion rules for context-dependent aggregation of structured news reports. *Journal of Applied Non-classical Logics*, 14:329–366, 2004.

[82] A. Hunter and R. Summerton. A knowledge-based approach to merging information. *Knowledge-Based Systems*, 19(8):647–674, 2006.

[83] A. Hunter and W. Liu. Fusion rules for merging uncertain information. *Information Fusion*, 7(1):97–134, 2006.

[84] H.E. Pence and A. Williams. ChemSpider: an online chemical information resource. *Journal of Chemical Education*, 87(11):1123–1124, 2010.

Index