Project-Team: LIS

Logical Information Systems
Theme Sym
Rennes —

November 2, 2009

Contents

1 Team

debug : Module named "composition" for the project LIS, section composition

Head of the team

Olivier Ridoux [Profession=Enseignant] [Category=UnivFr] [Professor, Université Rennes 1] [HDR=Habilite]

Administrative assistant

Lydie Mabil [Profession=Assistant] [Category=INRIA] [TR Inria]

Université Rennes 1 personnel

Sébastien Ferré [Profession=Enseignant] [Category=UnivFr] [Assistant Professor]

Annie Foret [Profession=Enseignant] [Category=UnivFr] [Assistant Professor]

Insa personnel

Mireille Ducassé [Profession=Enseignant] [Category=AutreEtablissementPublic] [Professor] [HDR=Habilite]

Véronique Abily [Profession=Technique] [Category=AutreEtablissementPublic] [Research Engineer (1/5 of the time)]

PhD students

Olivier Bedel [Profession=PhD] [Category=UnivFr] [Région Bretagne grant, since Oct. 2005]

Peggy Cellier [Profession=PhD] [Category=UnivFr] [MENRT grant, since Oct. 2005]

Associate members

Erwan Quesseveur [Profession=Enseignant] [Category=UnivFr] [Assistant Professor, Université Rennes 2]

François Le Prince [Profession=AutreCategorie] [Category=EtablissementPrive] [President of ALKANTE and Associate Professor, Université Rennes 2]

Visitors

Daniela Bargelli [Profession=Visiteur] [Category=UnivEtrangere] [Professor, MacGill University, Montreal]

2 Overall Objectives

2.1 Overview

debug : Module named "overview" for the project LIS, section presentation

The LIS team aims at developing *formal* methods for handling complex data sets in a *flexible* and *precise* way. "Flexible" means that the content determines the shape of the container. Very often, it is the opposite that is observed; e.g., the tree-like shape of a hierarchical file system enforces the tree-like shape of software packages. "Precise" means that any subset of the data set can be easily characterized. Again, it is the opposite that is often observed; e.g., in a hierarchical file system only sub-trees can be easily characterized. More and more information is available on the Web, and more and more information can be stored on a single machine. However, whereas the related low-level technology is developing, and performance is increasing, little is done for organizing the ever-growing amount of information. Therefore, the LIS team addresses the issues of organizing and querying information in general. The solutions are to be both formal and practical. Operational issues such as index technologies are important, but we are convinced that their scope is too limited to solve the crucial issues.

At a formal level, *queries* and *answers* are two key notions. It is nowadays standard to consider queries as logical formulas and answers as special models of queries. Computing the preferred model of a query in some context is conceptually easy, and it warrants flexibility. However, the opposite is not that easy in general; given a subset of the data, how can we compute a query of which it is a model? Given two different subsets of the data, how can we compute a query that explains the difference? Knowing this would warrant precision. The LIS team proved that *formal concept analysis* (FCA [?]) is a powerful framework for analyzing $\langle query, answer \rangle$ pairs. *Formal concepts* formalize the association between a query and its answers. Formal concepts are structured into a lattice which provides navigation links between concepts.

However, standard FCA cannot deal with queries considered as logical formulas (recall that this is the key for flexibility). Therefore a variant of FCA for logical description has been developed [**?**] altogether with the generic notion of *Logical information system* (LIS) that provided a reconstruction of all information system operations based on logical concept analysis. In particular, some data-mining operations are native in LIS [**?**, **?**, **?**].

The mottoes of the LIS research are:

1. *Never enforce* a priori *structure to information.* E.g., do not use hierarchical structures. Enforcing *a priori* structure causes the *tyranny of the dominant decomposition*[?]. For instance, the usual class-based organisation of source code makes highly visible the connections between methods of the same class, but masks the possible connections between methods in different classes.

   Instead, consider pieces of information as a bulk. Structure should emerge *a posteriori* from the contents or the point of view. As a consequence, updating the contents may change the structure: we accept it.

[?] *** ERROR: citation '**GanWil1999**' undefined ***

[?] *** ERROR: citation '**TOHS1999**' undefined ***

2. *Consider every possible rational classification, and permit changes at any time.* Here, rational means that what makes a piece of information belong or not to a class depends on the very piece of information, not on other pieces. The concept lattice induced by FCA is precisely a means to grasp all possible rational classifications.

3. *Rare events are as important as the frequent ones.* One cannot say *a priori* if a piece of information is interesting because it presents a frequent pattern, or because it presents a rare pattern.

   So, rare events must not be masked by statistical artefacts. Statistics is not forbidden, but it is only a complement of a symbolic logic approach.

4. *Queries should be possible answers.*

   In usual information systems (say relational databases or Web browsers) there is a strict dichotomy between queries (they are intensional expressions), and answers (they are strictly extensional expressions, i.e. sets of things). We contend that a good answer must be a mix of extensional and intensional answers. E.g. the good answer to "I would like to buy a book" is seldom the whole catalog of the bookshop; it is more relevant to answer such a query with other queries, like "Is this for a child" or "Do you prefer novels of documents".

   Note that hierarchical file systems already do that. Queries (i.e., *filepaths*) yield answers that contain other queries (i.e., *sub-directories*). One of the LIS achievements is a formalization of this behaviour that does not rely on an *a priori* hierarchical structure.

Our research is intended to be *vertical* in the sense that all aspects of information systems are of interest: design, implementation, and applications.

On the implementation side, the LIS team develops systems that present the LIS abstraction either at the file system level [**?**, **?**] or at the user level.

On the application side, the LIS team explores the application of LIS to *Geographical information systems* (GIS). The intuition here is that the traditional layered organization of information in GIS suffers a rigid structure of thematic layers. Moreover, GIS applications usually cope with highly heterogeneous information and large amount of data; this makes them an interesting challenge for LIS. The team also works on a data-mining interpretation of bug tracking. In this case, the intuition is that pieces of information relevant to software engineering, e.g. programs, specifications or tests, can be explored very systematically by a LIS. More generally, applications to software engineering are important for the team.

2.2 Key Issues

debug : Module named "key-issues" for the project LIS, section presentation

In its current state, LIS raises several questions that we wish to answer.

- The file system implementation of LIS can handle around 1 million elementary pieces of information.

  How can it handle more? Can we reach 100 million in the next few years?

- The LIS formalism is generic w.r.t. the logic used for describing pieces of information.

  What are the appropriate logics for the application fields that we have chosen? (GIS and error localization) Do we need a brand new logic for every application, or is there something that different applications can share?

- Genericity of LIS w.r.t. logic opens the door for creating ad hoc logics for describing pieces of information of an application. We already have proposed the framework of *logic functors* for helping a user build safely ad hoc logics. Logic functors are certified logic components that can be composed to form certified implementations of a logic.

  What are the useful logic functors? How can we be sure that a toolbox of logic functors is complete for a given purpose?

  Can the idea of certified composition be applied to another domain? Given a domain *foo*, *foo* functors would be certified *foo* components that can be assembled to form certified implementations of *foo* systems.

  Is it possible to certify other properties than meta-logical properties? E.g. is it possible to characterize complexity, or other non-functional properties like security?

- A family of non-commutative logics has developed over the years in the domain of computational linguistics, e.g. Lambek logic, pregroups. As for LIS, a great amount of creativity is expected for extending this family with ad hoc logics that would tackle fine-grained linguistics phenomena.

  Is it possible to build up an implementation of these logics using logic functors?

  Some LIS applications deal with objects that are sequential by nature (say, texts).

  Can these non-commutative logics primarily developed for computational linguistics help in LIS applications?

- Hierarchical file systems have a preferred metaphor which is the tree.

  What is the proper metaphor for LIS?

  The tree is also the graphical metaphor of hierarchical file systems.

  What is the graphical metaphor for LIS?

  Knowing this is crucial for the acceptance of LIS in end-user applications.

- Geographical information systems also suffer the *tyranny of the dominant decomposition*. Here, the dominant decomposition is in rigid thematic layers that inherit from plastic sheets of ancient map design. These layers are omnipresent in the design and interface of GIS applications.

  How can LIS abstract these layers, and still display layers when needed?

  Mining geographical information is difficult because of the layers and because it must cope with complex spatial relations.

  What is the proper modeling of these relations that will permit efficient LIS operations, including data-mining?