Empowering Ambivalence – Supporting multiple interpretations in knowledge-based systems

Pierre-Antoine Champin

Empowering Ambivalence

Supporting multiple interpretations in knowledge-based systems

Pierre-Antoine Champin

Habilitation à diriger des recherches

Université Claude Bernard Lyon 1

defended on 13/06/2017, before

- Deborah Estrin (Cornell Tech, USA)
- Fabien Gandon (INRIA Sophia Antipolis, FR)
- Dame Wendy Hall (University of Southampton, UK)
- Erik Mannens (Ghent Univsrity, BE)
- Alain Mille (Université Claude Bernard Lyon 1, FR)
- Barry Smyth (University College Dublin, IE)

Abstract

Building intelligent systems out of computers has been a continuous challenge for many computer scientists and developers. Among different paths to that goal, one that has been largely studied involves the explicit representation of *knowledge*, and the processing of those representations by generic reasoning engines. The advent of the Web, and then of mobile computing, has however dramatically changed the way we use computers, and with it our expectations of what such intelligent systems should be. It has also changed the means available to build them. The goal of this dissertation is to show how, in my work in the last ten years, I have been aiming at novel approaches to knowledge engineering, intending to tackle the new challenges and opportunities brought by the Web.

Knowledge-based AI has mostly developed on the premise that knowledge was *rare*, and as such should be made as *stable* as possible. A large part of our work has been trying to leverage the problems faced by any knowledge-based system when its context changes. Indeed, is not adaptability a core aspect of intelligence? But adaptive reasoning mechanisms must take into account, from the ground up, the *dynamics* of their knowledge base. It requires to embrace the fact that information is inherently *ambivalent*, that it acquires meaning (and hence becomes knowledge) only in the context of a particular problem or task. We have been pursuing a user-centered approach, where data collection and reasoning processes are as transparent as possible, and where meaning is not a pre-defined property of information, but negotiated and co-constructed with users.

I first present the theoretical framework that we have proposed to build knowledge-based systems exploiting *activity traces*. By capturing the inherent complexitiy of the user's task, this kind of knowledge allows for multiple interpretations, and hence requires a special kind of reasoning as well. Then I present a number of our works focusing on assisting a users in her task. One way is to simply present her with her traces in order to help her remembering them and sharing them with others. Another way is to use traces to detect failures and errors, and make helpful proposals for completing the task. The next chapter describes our activity related to Web technologies and Web standards. I show how the foundations of the Web accommodates and even encourages ambivalent information. As such, it allows to bridge the gap between documents, data and knowledge representations. In the next chapter, I focus on a specific class of Web documents, namely hypervideos. I present the models and tools we have proposed to process hypervideos, centered on the notion of annotation, and flexible enough to allow the emergence of new usages.

Finally, in the last chapter, to synthesize all the presented works, I propose the groundwork of a theoretical framework for knowledge representation, aimed to cope with, and account for multiple interpretations. In other words, it is an attempt to formalize ambivalent information and the dynamic reasoning processes that use them, allowing to build systems to adapt to the users, rather than forcing the users to adapt to them.

Résumé

La construction de systèmes intelligents a toujours été un défi important pour les chercheurs en informatique et les développeurs. Parmi les différentes voies envisagées pour atteindre ce but, celle de l'*ingénierie des connaissances* a été largement explorée, visant à représenter explicitement en machine les connaissances humaines, afin de pouvoir les traiter automatiquement à l'aide de moteurs de raisonnement génériques. Avec l'essor du Web, puis de l'informatique mobile, notre manière d'utiliser l'outil informatique a radicalement changé, et avec elle nos attentes à l'égard de ces systèmes intelligents. Les moyens disponibles pour construire de tels systèmes ont, eux aussi, largement évolué. L'objectif de ce mémoire est de montrer comment, au cours des dix dernières années, mon travail a visé à proposer des approches innovantes d'ingénierie des connaissances, afin de répondre aux nouveaux défis et opportunités amenés par le Web.

Les approches orientées connaissances en IA se sont principalement développées sous l'hypothèse que les connaissances sont *rares*, et que pour cette raison elles doivent être rendues aussi *stables* que possible. Une grande partie de nos travaux a consisté à répondre aux problèmes rencontrés par les systèmes à base de connaissances dont le contexte évolue. En effet, l'adaptabilité n'est-elle pas un aspect fondamental de l'intelligence ? Mais les mécanismes de raisonnement adaptatifs doivent, dès leur conception, prendre en compte la *dynamique* des bases de connaissances. Ceci suppose d'accepter le fait que toute information est intrinsèquement *ambivalente*, qu'elle n'acquiert de signification précise (et donc ne devient connaissance) que dans le contexte d'un problème ou d'une tâche spécifique. Nous avons suivi une approche centrée utilisateur, où la collecte des données et les processus de raisonnement sont aussi transparents que possible, et où la signification n'est pas une propriété pré-définie de l'information, mais au contraire négociée et co-construite avec les utilisateurs.

Je présente tout d'abord le cadre théorique que nous avons proposé pour construire des systèmes à base de connaissances exploitant les *traces d'activité*. En capturant la complexité inhérente aux tâches de l'utilisateur, ce type de connaissances se prête à de multiple interprétations, et nécessite donc de nouveaux types de raisonnement. Je présente ensuite un certain nombre de nos travaux visant à assister un utilisateur dans sa tâche. Une manière de le faire consiste à simplement lui présenter ses traces, afin de l'aider à se les remémorer ou à les partager avec d'autres. Une autre manière consiste à utiliser les traces pour détecter les échecs et les erreurs, afin de lui suggérer comment mener la tâche à bien. Le chapitre suivant décrit nos activités autour des technologies et des standards du Web. Je montre comment les fondations du Web supportent, voire même favorisent, l'ambivalence de l'information. À ce titre, elles permettent de créer un continuum entre documents, données, et représentations des connaissances. Dans le chapitre suivant, je m'attache à une classe particulière de documents Web, à savoir les hypervidéos. Je présente les modèles et outils que nous avons proposés pour traiter les hypervidéos, centrés sur la notion d'annotation, et suffisamment flexibles pour permettre l'émergence de nouveaux usages.

Pour finir, en vue de synthétiser tous les travaux présentés, le dernier chapitre propose les fondations d'un cadre théorique pour la représentation des connaissances, permettant de prendre en compte et de gérer de multiples interprétations. En d'autres termes, cette proposition tente de formaliser les informations ambivalentes et les processus de raisonnement dynamiques qui les utilisent, afin de permettre la construction de systèmes s'adaptant aux utilisateurs, plutôt que de forcer les utilisateurs à s'adapter à eux.

Acknowledgments

I would first like to thank the members of my jury, for accepting to make space in their schedule, which I know to be more than very busy, and honoring me with their names on the cover of this thesis. Their trust and recognition is an invaluable encouragement to carry on.

Marie-Pierre, Camil and Salomé, but also my parents

I is not another

Altough it is customary in a dissertation to ban “I” in favour of “we”, the reasons to do so are at best ambivalent, at worse ambiguous.

Many people were involved in the works related here (hopefully none of them were forgotten in the lines above), and it was natural for me to use the plural to report the results produced with them. However, I didn’t want to bind them by more personal claims or interpretations, for which I saw no reason to avoid the use of “I”. In the last chapter, on the other hand, the use of “we” is more an invitation for the reader to follow me in my reasoning. Still, although the proposal in that last chapter is a more personal one, it owes a lot to a many people, probably more than I even realize.

1. Introduction

Artificial intelligence (AI) can be argued to be as old (if not older) as computer science itself. Indeed, the question of intelligent machines was one of the motivations of Alan Turing (1950) for stating the principles of the Turing Machine, which remains until this day the abstract model of computers. Building intelligent systems out of computers has hence been a continuous challenge for many computer scientists and developers. Among different paths to that goal, one that has been largely studied involves the explicit representation of *knowledge*, and the processing of those representations by generic reasoning engines[1] (Schreiber et al. 2000).

The advent of the Web, and then of mobile computing, has however dramatically changed the way we use computers, and with it our expectations of what such intelligent systems should be. Of course, it has also changed the means available to build them. One could argue that the Web changes the fundamental assumptions on which AI was traditionally built, raising a number of new challenges, but also providing new opportunities. The goal of this dissertation is to show how, in my work in the last ten years, I have been aiming at novel approaches to knowledge engineering, intending to tackle those challenges and leverage those opportunities.

1.1. Dynamics and ambivalence

Since the first attempts to build knowledge-based computer systems, the process of acquiring and formalizing knowledge has been recognized as a major bottleneck in the building of such systems. With expert systems, knowledge was first acquired by knowledge engineers through interviews with domain experts. Such interviews are very time consuming, as the parties have different skills (respectively in formal models and in the application domain) and must learn enough from each other in order to reach an agreement on how to represent the experts' knowledge. Furthermore, a large part of the expert's knowledge is tacit (Nonaka and Takeuchi 1995) and eliciting it can be challenging. Finally, experts can be reluctant to disclose their knowledge if they have the feeling that the system is meant to replace them.

In order to tackle those difficulties, alternative approaches have been proposed, such as applying natural language processing (NLP) to a corpus of texts related to the application domain (Delannoy et al. 1993; Mooney and Bunescu 2005). The goal is to automatically or semi-automatically discover the domain terminology, and extract relevant knowledge in the desired formalism (rules, description logics, *etc.*). While less time-consuming than interviewing experts, such approaches produce knowledge representations that, most of the time, still require human validation. This is related to the fact that traditional AI is strongly rooted in formal logic, which leaves no room for approximate or relative truths[2]. Hence, whatever piece of knowledge that could be collected had to be scrutinized for validity and consistency. Not only does this make the building of the knowledge base costly, but also any evolution that this knowledge base might undergo. Knowledge-based AI has therefore mostly developed on the premise that knowledge was **rare**, and as such should be made as **stable** as possible, despite a few efforts to temper this tendency by applying agile methodologies (Auer and Herre 2007; Canals et al. 2013),

Another popular alternative approach is case-based reasoning (CBR), founded on a memory-model proposed by Schank (1982) and formalized by Aamodt and Plaza (1994). Schank points out that many reasoning tasks are not performed from first principles, but instead by reproducing or adapting the solution of a past similar problem. In CBR, problem solving knowledge is then captured by a set of *cases*, that are examples of previous problems with their solution. Reasoning is achieved by comparing the problem at hand with the ones previously solved, retrieved from the case base, and adapting the solution of the closest case. If successful, that adapted solution is in turn recorded as a new case[3]. One benefit of CBR over other approaches is that it does not require domain knowledge to be fully formalized. Instead, a representative set of prototypical cases is enough to get it started. Those may be significantly easier to acquire than more general knowledge, as they are not expected to hold a general or universal truth, but only a local solution in the context of the given problem. Another benefit is that a CBR system learns new cases as it is being used, which makes it able to improve over time.

Still, those benefits must not hide inherent difficulties. While the case base may be relatively easy to gather, it is only one of

the knowledge containers required by the CBR system (Richter 2003; Cordier 2008). Other knowledge containers include: the structural representation of cases, the similarity measure used to compare cases to the problem at hand, and the adaptation knowledge used to adapt its solution. Furthermore, an ever-growing case base requires continuous maintenance (Lopez De Mantaras et al. 2005, section 5; Cummins and Bridge 2011) in order to prevent pollution (from bad quality cases) or bloating (from an excessive quantity of cases). Maintenance becomes even more challenging when changes occur in the context in which the CBR system operates, and old cases become obsolete. Replacing obsolete cases by new ones is often not enough to take changes into account: all the knowledge containers mentioned above may have to evolve as well. The very structure of the cases (a problem and its solution) may have to be revised as the nature of the problem, or its understanding, may change over time. This of course may have a huge impact on the whole system, as all knowledge containers are strongly dependent on that structure.

A large part of our work has been inspired by CBR, and trying to leverage the problems faced by any knowledge-based system when its context changes. Indeed, is not adaptability a core aspect of intelligence? But evolving implies the continuous acquisition of knowledge. In other words, adaptive reasoning mechanisms must take into account, from the ground up, the **dynamics** of their knowledge base. It does not just mean being able to integrate new knowledge as it is being acquired; it does not even limit to being able to revoke old knowledge obsoleted by new information. It requires to embrace the fact that information is inherently **ambivalent**, that it acquires meaning (and hence becomes knowledge) only in the context of a particular problem or task[4]. Note that ambivalence must not be confused with *ambiguity*; effective ambivalence means that enough contextual information is available to disambiguate knowledge, to decide on a particular interpretation.

This is where the Web offers an unprecedented opportunity, as it has become the hub of most of our digital activities, even more, a large part of our personal and social lives. Its network structure naturally relates our actions to a vast amount of information, either text material (Wikipedia[5], various blogs or forums), structured databases (such as Freebase[6] or Musicrainz[7]), or interactive services (weather forecast, route planning, *etc.*). Of course, this is not new, and major Web companies have a long (and controversial[8]) history of studying and using this wealth of information, to gain deeper knowledge about their users and provide more targeted services. Their approach is however mostly one-way: users have few (if any) insight or control over the information that services have about them, or how it is used. And even if they did, that information is buried into machine-generated statistical models, that produce results mostly in a black-box fashion.

In contrast to this service-centered approach, we have been pursuing a user-centered approach, much in the line of Hsieh et al. (2013), where data collection and reasoning processes are as transparent as possible. It is important that every result can be explained and traced back to its premises, and that the interpretation choices about ambivalent information can be elicited. Indeed, the ultimate judge of the relevance of a reasoning process is the human on behalf of whom the system is working. It is therefore important that users have all the means to understand the results of the system, that they can comment on them, and that their feedback be collected as additional knowledge for future reasoning tasks. That way, meaning is not a pre-defined property of information, but negotiated and co-constructed with users. More generally, user feedback does not have to be explicit or direct: any interaction of the users with the system can be considered as a clue and participate to this negotiation.

1.2. Structure of the dissertation

The rest of this dissertation is structured as follows.

In Chapter 2, I will first present our work on building knowledge-based systems exploiting a special kind of knowledge, namely *activity traces*. More precisely, those systems keep track of how users have interacted with them in the past, in order to gather experience and, from this, to learn both *about* the users and *from* them. By capturing the inherent complexitiy of the user's task, this kind of knowledge allows for multiple interpretations, and hence requires a special kind of reasoning as well. I will describe the theoretical framework that we have proposed to build such trace-based systems, as well as the generic implementation of that framework which we have used in order to validate our proposals in various contexts.

One particular domain where experiential knowledge can prove useful is doubtlessly user assistance. In Chapter 3, I will present a number of our works focusing on that topic. The most straightforward use of traces to help users is simply to present them with their traces, in order to help them remember their activity or explain it to others. Of course, one's activity is not always a flawless and straight path, and it may be useful to detect failures and errors in the collected traces in order to make them more useful. Finally, observing the user's interactions with the system, and detecting unsuccessful or abnormal patterns, can help detect problems and make helpful proposals.

Chapter 4 will then describe our activity related to Web technologies and Web standards. The Web was initially designed as a document space, and documents are the traditional means for humans to represent their knowledge. Thus, *digital* documents, such as those used on the Web, can be designed in such a way that the knowledge they represent be equally usable by humans and machines. This is the starting idea that lead to the concept of the Semantic Web (Berners-Lee et al. 2001; Shadbolt, Hall, and Berners-Lee 2006). I will show how the REST architectural style, which is one of the foundations of the Web, accommodates and even encourages ambivalent information. As such, it allows to bridge the gap between documents, data and knowl-

edge representations.

In Chapter 5, I will focus on a specific class of documents, namely hypervideos. Indeed, while hypertext can build on centuries of practice around textual documents[9], video as a document form is hardly older than hypervideo itself, and lacks well established usages when it comes to active reading or annotation. Documentary structures for hypervideos must therefore be flexible enough to allow the emergence of new usages. Here again, ambivalence is a key to this flexibility. We have proposed models and tools to represent and process hypervideo, centered on the concept of *annotation*. Interestingly, video annotations share a lot of commonalities with the activity traces presented in Chapter 2: they both relate to something that is hard to grasp by computers (resp. the multimedia signal and the user's activity), which has an inherent temporal dimension to which both annotations and traces are anchored. Furthermore, video annotations are used, among other scenarios, to manually build traces of a recorded activity.

Finally, in the last chapter, to synthesize all the presented works, I will propose the groundwork of a theoretical framework for knowledge representation, aimed to cope with and account for multiple interpretations. In other words, it is an attempt to formalize ambivalent information and the dynamic reasoning processes that use them.

Notes

[1] In contrast, machine learning approaches aim at producing *models* from instance data. While those models can be used to make predictions or decisions, they arguably capture some knowledge about the domain. But they are usually very hard to interpret by humans, hence do not qualify as *explicit* knowledge representations.

[2] Alternative formalisms, such as modal logics (Chellas 1980) or fuzzy logics (Zadeh 1965) have been proposed, but with no real breakthrough in knowledge engineering.

[3] Actually, even a failed adaptation can be recorded in the case base, in order to prevent the system from making the same mistake again.

[4] CBR does not really meet this requirement, as all the knowledge containers, especially the structure of the cases themselves, are usually designed for a predefined class of problem.

[5] http://www.wikipedia.org/

[6] http://freebase.com/

[7] http://musicbrainz.org/

[8] See for example the controversy, reported by Arthur (2014), raised by Facebook's study (Kramer 2012) on users' emotions. More recently, Google's study on user's security questions (Bonneau et al. 2015) has also raised a few eyebrows.

[9] Note that this can be both an advantage and a hindrance, as old habits may impede the emergence of new practices.

Chapter bibliography

Aamodt, Agnar, and Enric Plaza. 1994. "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches." *AI Communications* 7 (1): 39–59.

Arthur, Charles. 2014. "Facebook Emotion Study Breached Ethical Guidelines, Researchers Say." The Guardian. June 30, 2014. http://www.theguardian.com/technology/2014/jun/30/facebook-emotion-study-breached-ethical-guidelines-researchers-say.

Auer, Sören, and Heinrich Herre. 2007. "RapidOWL — An Agile Knowledge Engineering Methodology." In *Perspectives of Systems Informatics*, edited by Irina Virbitskaite and Andrei Voronkov, 4378:424–30. LNCS. Springer. http://www.springerlink.com.gate6.inist.fr/content/5l6r104080127u34/abstract/.

Berners-Lee, Tim, James Hendler, Ora Lassila, and others. 2001. "The Semantic Web." *Scientific American* 284 (5): 28–37.

Bonneau, Joseph, Elie Bursztein, Ilan Caron, Rob Jackson, and Mike Williamson. 2015. "Secrets, Lies, and Account Recovery: Lessons from the Use of Personal Knowledge Questions at Google." In *24th International Conference on World Wide Web*, 141–150. Florence, Italy: ACM. http://dl.acm.org/citation.cfm?id=2736277.2741691.

Canals, Gérôme, Amélie Cordier, Emmanuel Desmontils, Laura Infante-Blanco, and Emmanuel Nauer. 2013. "Collaborative Knowledge Acquisition under Control of a Non-Regression Test System." In *Workshop on Semantic Web Collaborative Spaces*. Montpellier, France. https://hal.archives-ouvertes.fr/hal-00880347.

Chellas, Brian F. 1980. *Modal Logic: An Introduction*. Cambridge [Eng.] ; New York: Cambridge University Press.

Cordier, Amélie. 2008. "Interactive and Opportunistic Knowledge Acquisition in Case-Based Reasoning." Thèse de Doctorat en Informatique, Université Lyon 1. http://liris.cnrs.fr/publis/?id=3776.

Cummins, Lisa, and Derek Bridge. 2011. "Choosing a Case Base Maintenance Algorithm Using a Meta-Case Base." In *Research and Development in Intelligent Systems XXVIII*, 167–180. Springer. http://link.springer.com/chapter/10.1007/978-1-4471-2318-7_12.

Delannoy, J. F., C. Feng, S. Matwin, and S. Szpakowicz. 1993. "Knowledge Extraction from Text: Machine Learning for Text-to-Rule Translation." In *Proceedings of European Conference on Machine Learning Workshop on Machine Learning and Text*

Analysis, 7–13. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.8376&rep=rep1&type=pdf.

Hsieh, Cheng-Kang, Hongsuda Tangmunarunkit, Faisal Alquaddoomi, John Jenkins, Jinha Kang, Cameron Ketcham, Brent Longstaff, et al. 2013. “Lifestreams: A Modular Sense-Making Toolset for Identifying Important Patterns from Everyday Life.” In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, 5:1–5:13. SenSys ’13. New York, NY, USA: ACM. https://doi.org/10.1145/2517351.2517368.

Kramer, Adam D.I. 2012. “The Spread of Emotion via Facebook.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 767–770. CHI ’12. New York, NY, USA: ACM. https://doi.org/10.1145/2207676.2207787.

Lopez De Mantaras, Ramon, David Mcsherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, et al. 2005. “Retrieval, Reuse, Revision and Retention in Case-Based Reasoning.” *The Knowledge Engineering Review* 20 (03): 215–240. https://doi.org/10.1017/S0269888906000646.

Mooney, Raymond J., and Razvan Bunescu. 2005. “Mining Knowledge from Text Using Information Extraction.” *SIGKDD Explor. Newsl.* 7 (1): 3–10. https://doi.org/10.1145/1089815.1089817.

Nonaka, Ikujiro, and Hirotaka Takeuchi. 1995. *The Knowledge Creating Company*. Oxford University Press, Oxford (GB).

Richter, Michael M. 2003. “Knowledge Containers.” *Readings in Case-Based Reasoning*. http://pages.cpsc.ucalgary.ca/~mrichter/Papers/Knowledge%20Containers.pdf.

Schank, R.C. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Vol. 240. Cambridge University Press Cambridge.

Schreiber, Guus, Hans Akkermans, Anjo Anjevierden, Robert de Hoog, Nigel Shadbolt, Walter Van de Velde, and Bob Wielinga. 2000. *Knowledge Engineering and Management: The CommonKADS Methodology*. MIT Press.

Shadbolt, Nigel, Wendy Hall, and Tim Berners-Lee. 2006. “The Semantic Web Revisited.” *Intelligent Systems, IEEE* 21 (3): 96–101.

Turing, Alan M. 1950. “Computing Machinery and Intelligence.” *Mind*, 433–460.

Zadeh, L.A. 1965. “Fuzzy Sets.” *Information and Control* 8 (3): 338–53. https://doi.org/10.1016/S0019-9958(65)90241-X.

2. Experiential knowledge and trace-based systems

This chapter is mostly based on the papers by Champin, Mille, and Prié (2013) and Cordier et al. (2013), and presents a synthesis of our work on modeling experiential knowledge and reasoning with it.

2.1. Motivation

By design, computers continuously produce and use traces. Every computational process works with data stored in more or less volatile memories, and produces new data that is in turn stored in those memories. Every digital inscription is therefore by definition a trace of the processes that allowed it to be produced. Those digital inscriptions account for computational processes, insofar as they result from the execution of programs (Deransart, Ducassé, and Langevine 2002), but they also account for the interactive processes between the human and the machine, insofar as they pertain to computer applications that are used by humans, in the context of their activity which is *mediated* by the computer (Kaptelinin 1996).

A computer-mediated activity produces traces linked to every computational process taking part in that activity. For example, a typical workday on a computer connected to the World Wide Web would produce different kinds of traces both on that computer and on other involved machines. Should we consider the resulting traces at the end of that day, we would have all documents, created or received, e-mail and instant messages, browsing history, to name only those inscriptions handled by the user. But we would also have the log files of all involved applications and servers. Should we be interested in the traces related to the proceeding of the activity, we would add different data structures handled by those applications: messages being written, open windows, load indicators for CPU, memory or network, *etc*.

Those traces are digital traces in the broad sense (Laflaquiere et al. 2006), that can be of any kind: document, data structure, log file, *etc*. We can make two remarks about them. First, their status of trace is only marginally taken into account by computer systems, and the interpretation of such inscriptions as traces is usually performed outside the system that produced them. Consider for example a contact in an address book, interpreted as a trace of the activity resulting in its recording. The trace status of those inscription is nevertheless acknowledged through the recording of temporal information regarding the processes producing and altering these inscriptions, *e.g.* the creation and modification dates of a file, or the timestamp of each entry in a log file. Second, each application handling traces as such has its own dedicated models and formats for representing traces, for example Learning Management Systems (George, Michel, and Ollagnier-Beldame 2013) or browser history. Despite some recents efforts to unify digital traces in some domains, such as social applications (Snell and Prodromou 2016), health (Hsieh et al. 2013), or data provenance (Moreau and Missier 2013), there is not yet a general model that would allow a cross-application and cross-domain use of digital traces, and provide generic processes for manipulating them.

Our goal, as ambitious as it may seem, is however to make traces a first-class citizen of computer systems. That way, we aim at capturing an important, and often overlooked, kind of knowledge: the *experience* that result from remembering and reusing past activities. We need to define a new *digital trace* object, which includes all the features (especially temporal ones) allowing it to be explicitly treated as a trace by applications, while remaining generic enough to be usable across various application domains.

In this chapter, we present the knowledge-engineering approach to digital traces that we have been developing for a number of years. This approach aims at building modeled-trace based systems (MTBS). Modeled-traces (or m-traces for short) are made of timestamped elements named *obsels* (contraction of “observed element”) and are associated with a *trace-model*. The trace-model provides a guideline for building and interpreting the m-trace. Computations on m-traces are most of the time *transformations* into new m-traces, that can be seen as a form of automated interpretation of the source m-trace.

2.2. A knowledge based approach for modeling and transforming digital traces

Ad-hoc uses of digital traces for observing

Observing an activity in order to understand it consists in collecting observable elements related to that activity, in order to build *evidences* guiding an interpretation. The sequence of evidences forming a trace can therefore be used to support, justify and explain interpretations. When the activity is computer-mediated, it is relatively easy to instrument the computer environment so that it collects digital traces made of potentially meaningful elements.

Digital traces were first used to ease the debugging of computer programs, with the idea that a knowledgeable observer (usually the programmer) could analyze the collected observations and interpret those traces, in order to understand the program's behavior and fix it if needed. Computer systems have long been able to produce a memory dump whenever an exception[1] is raised during the execution of a program. The produced trace can be completely standard or customized by analysts, who can set up tracing tools in order to follow only the elements that they deem relevant (Deransart, Ducassé, and Langevine 2002).

Like programmers, who can analyze the behavior of a computational process they designed, one can observe computer-mediated processes or activity as soon as the environment is instrumented in order to leave persistent traces. Such an analyst can be a professional one, or simply somebody willing to review or understand that activity—possibly the very person having performed that activity.

Then they need an interpretable representation of the collected traces. Such representations are always the result of a computation, either elementary (*e.g.* the hexadecimal representation of a memory dump) or more complex (*e.g.* a histogram of the time spent on that activity per day). A statistical processing of those elements can also be performed, using heuristics that depend on the purposed interpretation. A notable example is digital trace mining, which seeks to detect structural recurring patterns, in order to identify relevant behaviors or processes (Song, Günther, and Van der Aalst 2009; Van der Aalst et al. 2003; Cook and Wolf 1998). An important and well established use case of those techniques is to provide recommendations and personalization to the users of the traced system. This is applied in various contexts, such as Learning Management Systems (Marty, Carron, and Heraud 2009) or web sites[2] (Sachan et al. 2012).

While trace mining is usually associated with big-data, and the analysis of trends among a large population of users, digital traces may prove valuable at a much smaller scale, namely that of the individual traced user. Deborah Estrin (2014) captures this idea with the concept of "small data, where n=me", advocating for the extensive use of personal traces for providing insight on one's behaviour (implying, among other things, the right for users to access their traces collected by third-party applications).

Whatever the techniques or the scale of trace analysis, the analyst has a fundamental role to play in the process of knowledge discovery. The use of sequential interaction traces has been studied by Fisher and Sanderson (1996) who showed that the crucial task for the analyst is to find which transformations to apply to the raw observations in order to discover useful descriptions for explaining the observed process. More recently, Amer-Yahia et al. (2014) have proposed a formal algebra to describe those transformations, for preparing data produced by social applications before applying data mining techniques. Indeed, raw observations are piecemeal and expressed from the perspective of collecting devices, *i.e.* in a low level register. Knowledge, on the other hand, is expressed in the register of the activity. Hence the need, to interpret traces, for a transformation carrying the skills and knowledge of the analyst, so as to rephrase sequences of raw observations into sequences of meaningful activity elements.

The research works and practices described above suggest that collecting, modeling, transforming, rephrasing and interactively exploring are necessary steps whenever the observation requires multiple interpretations. We have proposed a unifying approach in order to integrate those steps with a rich representation structure dedicated to observation traces.

Modeling digital traces: associating an interpretation model to observed elements

In numerous applications processing traces and sequential data, the semantics of those data is mostly implicit. Even when documented, it is often loosely defined, reducing developers and analysts to a hazardous guesswork based on data labels and sample values. The outcome of knowledge discovery processes could be used to improve this situation, but it can not in general be reliably attached to the original traces, as their format is not designed to allow such linking[3].

We propose a new perspective on traces, considering them as *knowledge inscriptions* meant to carry not only the collected information, but also the elements allowing their interpretation by humans as well as computers. This brings traces into the domain of knowledge engineering, thus significantly widening the range of available tools for processing, transforming, sharing and reusing them, thanks to an explicit and operational semantics. We choose to express this semantics as a *trace-model* asso-

ciated to the set of observed elements, and playing three roles. First, it plays the role of a *vocabulary* used to describe the observed elements, unambiguously relating them to the model. Second, it plays the role of a *schema*, constraining the structure of the observed elements. As such, it can be used to distinguish valid (or consistent) observations from invalid ones. Third, it plays the role of an *ontology* (Bachimont 2004, p.160-), allowing to infer new information from what has been actually observed.

But obviously, a unique model can not be sufficient to describe all computer-mediated activities, not to mention the multiplicity of perspectives on a given activity. We therefore propose a generic *meta-model* specifying how trace-models can be described, which will be described in Section 2.3.

We consider the interpretation of a trace, expressed using an initial trace-model, as a "rephrasing" of that trace into another model, working at a different level of abstraction. For example, it would seem natural to interpret the sequence `[click icon foo], [word processor starting], [foo loading], [window displayed]` as the user opening a document named "foo". Hence, that sequence could be rephrased into a single observarion `[open document foo]`. The observed element in this new trace belongs to a new trace-model, which has a higher level of abstraction than the one of the initial sequence.

2.3. Modeled-trace based systems

I now present the meta-model that we have proposed for representing and processing m-traces in dedicated knowledge-based systems, MTBSs.

> **Example:** In the rest of this chapter, I will illustrate the presented notions with the following running example: Alice uses an e-mail application to communicate and exchange documents with her colleagues.

Modeled-trace

The central notion of our meta-model is that of *modeled-trace* (m-trace), but we first need to define the notions of *obsel* and *trace-model*.

Every traced activity is represented by a list of *observed elements* or **obsels**. This neologism is inspired by the word "pixel" (picture element), and was coined to insist on the fact that the content of any trace is the result of an observation, hence unavoidably biased[4]. Every obsel has:

- a begin time-stamp and an end time-stamp, anchoring the obsel in the time of the activity; both time-stamps can be equal, in the case of an *instantaneous* observation;
- a type, associating this particular obsel to an explicit category from the trace-model;
- a set of attributes, of the form <attribute-type, value>.

Let us note that the components of an obsel are, on purpose, only loosely specified by the meta-model. They are highly dependent on the represented activity, which should therefore be described by a **trace-model**. That model must specify:

- how *time* is to be represented (simply a time unit, as discussed in the next subsection "Representing time");
- the obsel types that can be used to describe the activity;
- for each obsel type, which attribute types can be used, and what type of value they may have;
- a set of binary relation types that may exist between obsels;
- a set of integrity constraints that an m-trace and its obsels must satisfy to comply with this trace-model.

> **Example:** In Alice's "e-mail" activity, we decide to measure time to the second.
>
> There are three obsel types: the receiving of a message (`RecvMsg`), the sending of a message (`SendMsg`) and the saving of an attachment (`SaveAtt`). Obsels of types `RecvMsg` and `SendMsg` have two attributes in common: the content of the message, and the content of their attachment if any. Moreover, obsels of type `RecvMsg` have an extra attribute holding the e-mail address of the sender, while obsels of type `SendMsg` have an attribute holding the e-mail address(es) of the recipient(s) of the message, and one holding the path of the attached file, if any. Finally, obsels of type `SaveAtt` have an attribute holding the name under which the attachment was saved.
>
> The trace-model also defines three relation types. The first two, `RepliesTo` and `Forwards`, both link an obsel of type `SendMsg` to one of type `RecvMsg`, to indicate that the sent message was, respectively, a response to the received message, or its forwarding to another recipient. The third relation type, `From`, links an obsel of type `SaveAtt` to one of type `RecvMsg` to indicate which message the saved attachment came from.

This trace-model constrains all obsels to be instantaneous, *i.e.* to have the same begin and end times-stamps. Furthermore, the second member of a `From` relation must have an attachment, *i.e.* the corresponding attribute must not be empty. Finally, in a `SendMsg` obsel, the two attributes holding the attachment content and its file-name must be either both empty or both non-empty.

An obsel type in a trace-model can also be associated to one or more parent type(s). This relationship has the standard subclass semantics (also called "a kind of"), and is interesting at several levels. At the syntax level, it allows the children types to inherit attribute definitions from their parent types, and encourages modularity in the design of the trace-model. At the semantic level, it implies that all obsels of the children types will also belong to the parent types, enabling more reasoning (and hence transformations) on the m-traces. Relation types can also have parent relation types.

Example: In our trace-model above, the common attributes of `RecvMsg` and `SendMsg` can be moved up in a parent type, which we can call `Message`. The resulting trace-model is represented in Fig. 2.1.

Finally, a trace-model can be linked to a number of parent trace-models, provided that they all share the same representation of time. In that case, the child trace-model will inherit all obsel types, attribute types, relation types and integrity constraints of all its parents. This is valuable from a knowledge engineering perspective, as it encourages the reuse of previously defined trace-models, together with the reasoning processes and transformations associated with them.

Example: The "e-mail" trace-model described above could be inherited by a broader trace-model, also inheriting a trace-model for "word processing", providing a more holistic view on Alice's (or any office worker's) activity. Another trace-model, dedicated to a specific e-mail application, could also inherit our example trace-model, and extend it with functionalities that are specific to this application (*e.g.* contact management, message folders...).

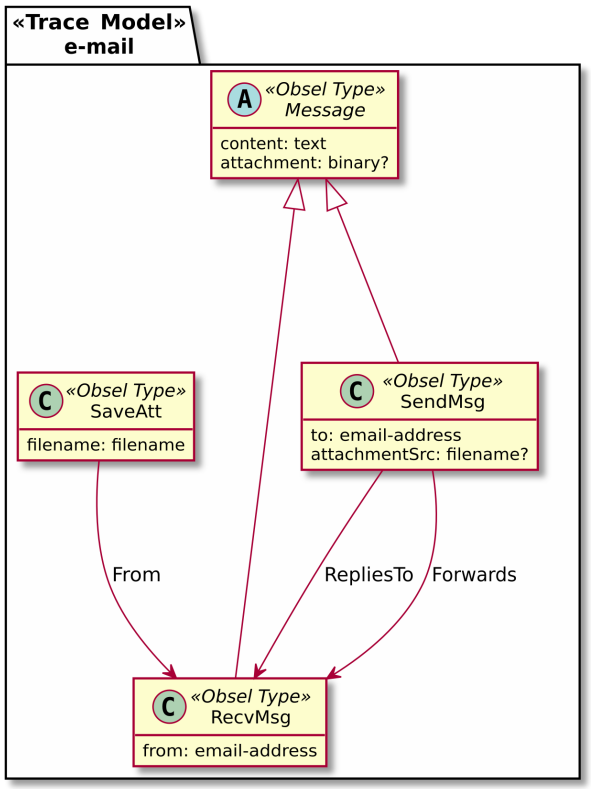


Fig. 2.1 An example trace-model (Champin, Mille, and Prié 2013).

We are now ready to precisely define a **modeled-trace**. It is specified by:

- a reference to a trace-model,
- a time interval called the *temporal extension* of the trace,
- a set of obsels,
- a set of typed binary relations between those obsels.

The temporal extension is the period of time during which the traced activity was recorded. While the obsels of the m-trace must all be between the bounds of the temporal extension, the time-stamp of the first and last obsel may not match exactly these bounds. Indeed, the *absence* of obsels, in some parts of the temporal extension, may be relevant for interpreting the trace.

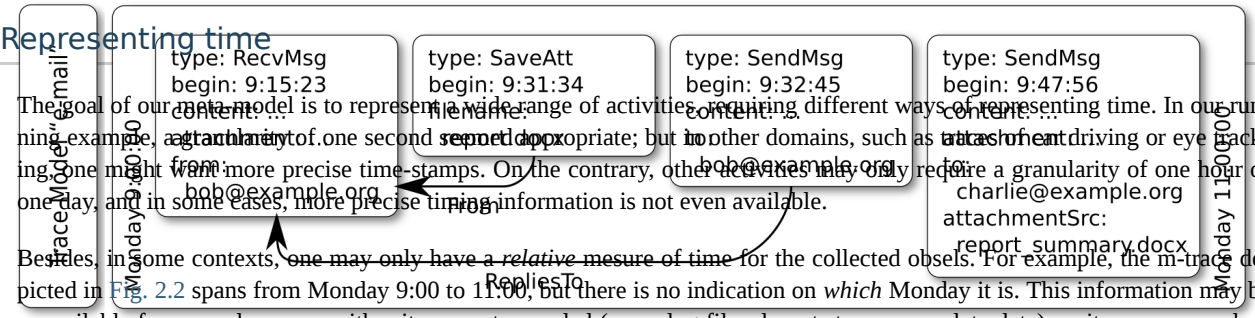
The temporal extension is described using the time representation specified by the trace-model. Of course, the obsels and their relations are also described accordingly to the trace-model.

Example: Fig. 2.2 shows an m-trace representing Alice's e-mail activity. It refers to the "e-mail" trace-model described above. Its temporal extension spans from Monday 9:00 AM to 11:00 AM.

It is composed of four obsels. To keep it simple, we have not represented the end time-stamps (as they are always equal to the begin time-stamp). At 9:15, Alice receives an e-mail from Bob. At 9:31, she saves the attached file as `report.docx`, then replies to Bob at 9:32. At 9:47, she sends a message to Charlie, attaching a file named `report-summary.docx`.

Representing time

The goal of our meta-model is to represent a wide range of activities, requiring different ways of representing time. In our running example, a granularity of one second is appropriate; but in other domains, such as track of car driving or eye tracking, one might want more precise time-stamps. On the contrary, other activities may only require a granularity of one hour or one day, and in some cases, more precise timing information is not even available.



Besides, in some contexts, one may only have a *relative* mesure of time for the collected obsels. For example, the m-trace depicted in Fig. 2.2 spans from Monday 9:00 to 11:00, but there is no indication on *which* Monday it is. This information may be unavailable for several reasons: either it was not recorded (some log files do not store a complete date), or it was removed on purpose, for example for privacy reasons. In other contexts, the temporal information may be even scanter, obsels being merely ordered in a sequence.

To account for all those situations, our meta-model requires that:

- per its definition, a trace-model specifies a time unit u;
- every m-trace has an origin o (see below);
- every time-stamps in an m-trace (its temporal extension and its obsels) is an integer t, representing the instant $o + tu$.

The **origin** is a character string. If it is a standard representation of an instant, *e.g.* using the RFC 3339 format (Klyne and Newman 2002), at least as precise as the unit of the trace-model, then the temporal extension and the obsels can be absolutely dated. Their time-stamps can be converted to other time formats, and compared with any other absolute time-stamp[5]. On the other hand, an origin not complying with a standard format is called an **opaque origin**. The time-stamps of the corresponding trace can be compared with each other, but not with any arbitrary other time-stamp. Note however that an opaque origin is assumed to always identify the same instant, so if two m-traces have the same opaque origin, their time-stamps are assumed to be comparable[6]. As most transformations do not alter time-stamps, they usually preserve the origin of the m-trace, making the source and the transformed trace comparable with each other.

> **Example:** The trace from Fig. 2.2 must be represented with an opaque origin, as we do not know on which Monday it was recorded. We chose to keep "Monday" in the origin to provide a hint to users. The temporal extension spans from 32400 (*i.e.* 9 hours) to 39600 (*i.e.* 11 hours). All time-stamps of the obsels are converted accordingly. The resulting m-trace is depicted in Fig. 2.3.

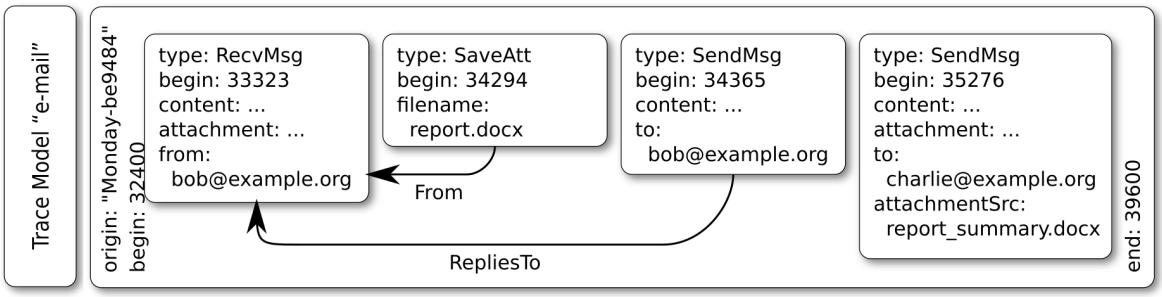


Fig. 2.3 The example from Fig. 2.2, with a unified representation of time (Champin, Mille, and Prié 2013).

Finally, to represent a sheer ordered sequence of obsels without any quantifiable temporal information, we define the special time unit `sequence`. This unit imposes the following constraints:

- the origin of the m-trace must be opaque;
- every obsel must have equal begin and end time-stamps, and all obsels of the m-trace must have different time-stamps;
- only the *order* of the time-stamps is significant; their absolute value gives no information of duration. One can not assume, for example, that the duration between time-stamps 1 and 2 is the same as between 2 and 3.

This allows to handle cases where the only information about the obsels is a total ordering. Other special units could be proposed to handle other kinds of limited temporal information.

Architecture of an MTBS

We are now ready to describe the overall architecture of an MTBS, illustrated in Fig. 2.4.

The core component of such a system is the **modeled-trace management system** (MTMS). It plays a similar role to that of a the database management system in a standard application, but manages instead m-traces complying with the meta-model presented above. It must be flexible enough to allow several trace-models to coexist (and evolve). It must also support the intrinsic dynamics of traces. Finally, it must be able to handle modeled-trace transformations (that will be discussed in more detail in Section 2.4).

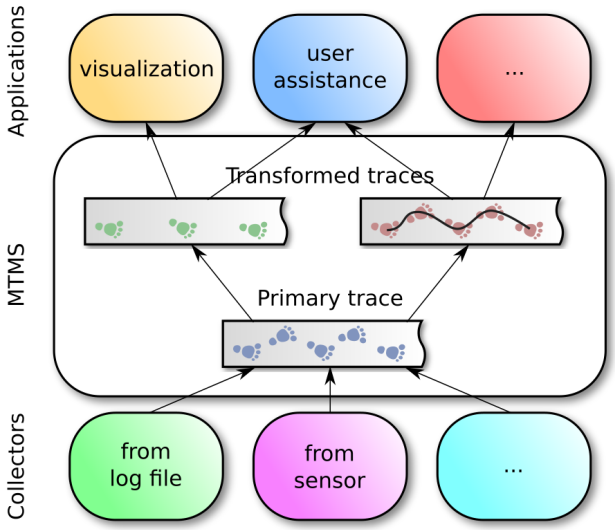


Fig. 2.4 General architecture of a MTBS built around a MTMS (Champin, Mille, and Prié 2013)

The MTMS is fed by a number of **collectors**, whose role is to gather the information required to build one or several m-traces. That information can be gathered synchronously, by observing the traced activity while it is taking place, or *a posteriori*, for example by examining log files. The trace-model of the collected m-trace determines which part of the available information is kept, and how it is organized to constitute the obsels of the m-trace[7]. Any m-trace produced by a collector is called a **primary trace**, as opposed to the *transformed traces* that are computed by the MTMS from other m-traces (either primary or transformed).

Finally, all m-traces can be used by application modules. Some of them can be used to display m-traces to the user in different ways, either very generic (a table listing all the obsels) or specific to a given trace-model, or even to a specific task. Other modules will process the m-traces in order to alter their own behavior, such as assistance system reusing past experiences of the user.

2.4. Trace based reasoning

Transformed m-traces

Most of the time, primary traces are not directly (or easily) usable by application modules; it is necessary to pre-process and transform them. One of the key roles of the MTMS is to perform those transformations, in order to support multiple interpretations and reasoning with m-traces.

A **transformed trace** is specified by:

- one or more *source* m-traces (which can be either primary or transformed),
- a reference to a transformation method,
- optionally one or more parameters influencing the execution of the transformation method.

All the properties of the transformed trace (its model, its temporal extension, its obsels and their relations) are deterministically computed by the transformation method, provided with the source traces and the parameters. Also, note that transformations can be chained (as the sources of a transformed trace can be transformed traces themselves), in order to produce complex workflows.

While the range of possible transformation methods is very large, we can distinguish three main classes of elementary methods.

- *Selection* methods keep only a subset of the obsels of a unique source trace, with respect to a given criterion. The model of the transformed trace is usually the same as the source trace, as well as the temporal extension (unless the criterion is about time-stamps).

  > **Example:** In the “e-mail” trace-model of our running example, the following selections can be considered: keep only obsels between 9:30 and 9:40 (temporal criterion), keep only obsels of type `SendMsg` (typology criterion), keep only obsels with an non-empty attachment (attribute criterion), keep only obsels that have been replied to (relation criterion).

- *Fusion* methods gather in the transformed trace all obsels from several source m-traces. If the sources have different trace-models, the model of the transformed trace should inherit all their model (which implies that they have the same

representation of time).

> **Example:** We could combine the trace of our running example with another of Alice's traces, also complying with the "e-mail" trace-model, but covering the period between 11:00 and 13:00 that same day, to analyze a longer part of her activity. We could also merge her trace with the "e-mail" trace from Bob at the same time, in order to study more precisely how the two of them communicate. Finally, we could combine that trace with Alice's "word processing" trace, to analyse her office activity in a larger context. That larger context could in particular provide insight on the "e-mail" part of the activity, for example by showing that `report-summary.docx` is a modified version of `report.docx`.

- *Rewriting* methods populate the transformed trace with new obsels, that are derived from the obsels of a unique source trace. It may consist in copying those obsels with less information (removing or altering some of their attributes) or more (inferring new attributes or relations from the content of the source trace or from external knowledge). But rewriting is not necessarily injective; obsels in the transformed trace may be derived from several source obsels, collectively satisfying a number of constraints.

> **Example:** A trace complying with the "e-mail" trace-model can be anonymized by removing all sender and recipient attributes[8]. On the other hand, we could imagine to enrich the source trace by tagging obsels with an emotion detection algorithm (which would require to extend the original trace-model). Another rewriting could consist in summarizing e-mail activity, by generating one obsel per day, its attributes indicating the number of sent and received messages (this would of course require a dedicated trace-model, different from the one presented earlier). Finally, we can imagine a more elaborate kind of summary, where a sequence of messages replying to each other would be rewritten into a single obsel of type `Conversation`, while a sequence of sent messages with the same content to different recipients would be rewritten into a single obsel of type `Broadcast`.

Note that rewriting transformations may apply not only to obsel attributes, but also to their time-stamps, as well as those of the m-trace. Reducing their precision or changing an absolute origin to an opaque one may be necessary to efficiently anonymise the m-trace. It could also be used to align two m-traces originally captured at different times, in order to compare them. For example, one may want to compare the execution of the same task in two different contexts.

Fig. 2.5 illustrates those notions. It also points out that not only are transformed traces linked to their source trace, but every obsel in a transformed trace can keep track of its corresponding source obsels. Thus, any obsel at any level can be *explained* by the process (transformation methods) and the data (source obsels) from which it was produced.

Reasoning with transformations

A transformation chain, such as the one depicted in Fig. 2.5, can arguably be considered as a reasoning process. It involves different knowledge containers: the factual knowledge contained in the primary traces, the structural knowledge contained in the various trace-models, and the inferential knowledge contained in the different transformation methods. Moreover, that particular arrangement of transformations (with the parameters of each transformed trace) also carries some knowledge which can be either general (*e.g.* `SaveAtt` obsels are not relevant for rewriting to an "e-mail summary", so they can be filtered out) or specific to a given context (*e.g.* those two primary traces are related to each other, so they should be merged).

We have proposed (Cordier et al. 2013) that trace based reasoning (TBR) can be structured as an interactive cycle of three steps – inspired by the CBR cycle described by Aamodt and Plaza (1994).

- The *elaboration* step consists in setting up the transformation chain that is relevant to solve the problem at hand. This usually amounts to identifying in m-traces reusable *episodes*, *i.e.* sets of obsels that meet a number of criteria; those episodes will typically appear as aggregate obsels in a transformed trace (such as the `Conversation` and `Broadcast` obsels in Fig. 2.5). For the classes of problems anticipated by the MTBS designers, the appropriate transformations will be provided with the system. However, nothing prevents users to add their own transformation to answer unanticipated questions. Indeed, MTMSs can handle multiple concurrent transformations of the same m-trace, in order to support multiple (and sometimes contradictory) interpretations of the primary traces.
- In the *retrieval* steps, the MTMS executes the transformations specified before. Depending on the kind of transformation, it can be submitted by the user to a number of constraints: on the number of episodes to retrieve, on the search algorithm to use, on the minimum certainty degree to apply... Note that, contrarily to the cases in traditional CBR, the episodes in TBR are never isolated as self-sufficient structures, but remain linked to the original obsels. They are always part of a "bigger picture", and their context of occurrence can always be tapped whenever their content itself is not enough to decide on the most relevant episode(s) to retrieve.
- The *reuse* step is when the retrieved episodes are effectively used (possibly with some adaptation) to solve the problem at hand. This can be done in various different ways, typically outside the MTMS itself (in the application modules of

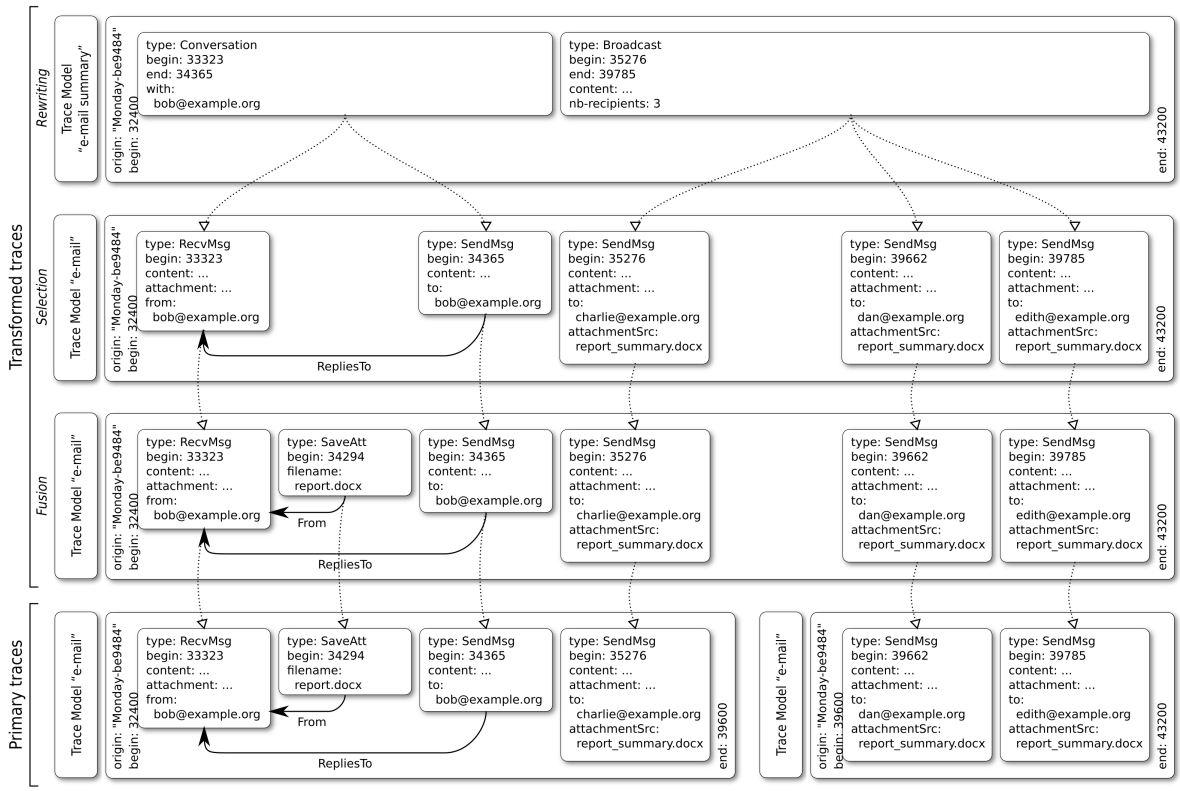


Fig. 2.5 Transformed traces.

This figure shows how the three kinds of transformations can be applied in our running example. The first (from the bottom) transformed trace is a fusion of the two primary traces, which represent the same "e-mail" activity at different periods of time. The second transformed trace is a selection, keeping only obsels of type `Message` (recall that `RecvMsg` and `SendMsg` both inherit that obsel type). The third transformed trace is a rewriting into a more synthetic trace-model, classifying sequences of obsels into different communication patterns. Note how each transformed obsel is linked to one or more source obsels (dotted arrows).

Fig. 2.4). Whatever the functions of those application modules are, they are integrated in the user's traced activity, and therefore fed back to the MTMS. This is what closes the cycle, even if we do not have an explicit "retain" phase as in the CBR cycle. Indeed, Ollagnier-Beldame (2011) and Terrat (2015) have shown that even the simple fact of displaying the m-trace to the user can help improve their appropriation of the system, an effect called reflexivity. Of course, knowledge extraction can also be the explicit goal of that step, either for an external analyst or as an advanced form of reflexivity. This has been studied by Mathern et al. (2012) and Barazzutti, Cordier, and Fuchs (2015).

The user is therefore at the center of the cycle, being strongly involved in each step. Knowledge is dynamically **co-constructed**, the system sustaining pre-defined interpretations and providing reflexivity, and the user assessing those interpretations in *context*, and testing new ones when the former are not satisfactory. As such, the system can be continuously adapting to changes.

Of course, to support this level of interactivity, MTBSs must be equipped with intelligible user interfaces, both for presenting m-traces and for designing new transformation. Studying such interfaces, in order to determine which features make them efficient, is still an open question and probably one of the key points in the future development of MTBSs, which we have started to investigate (Besnaci, Guin, and Champin 2015; Kong Win Chang et al. 2015).

An open-source reference implementation

The meta-model presented in this chapter is the result of many discussions and iterations. Since 2009, we have been working on a reference implementation, whose first goal was to help stabilize the meta-model (as many problems only appear with a concrete use cases). Its second goal was to ease and speed up the development of experiments aiming at validating and/or extending the meta-model. This implementation is named kTBS (a kernel for trace based systems), and is available at http://tbs-platform.org/ktbs.

kTBS is open-source, in order to foster its reuse both in and outside our research group. It is designed as a RESTful Web ser-

vice (Fielding 2000), in order to be easily integrated with other systems, regardless of their own architecture or programming language[9]. Internally, it stores all its data using the RDF data model (Schreiber and Raimond 2014), which meets the requirements of flexibility of our meta-model. RDF also comes with a powerful query language (Harris and Seaborne 2013), and expressive ontology languages (Hitzler et al. 2009). Externally, kTBS exposes and consumes data in the JSON-LD format (Sporny, Kellogg, and Lanthaler 2014). As stated above, the next step is to provide kTBS with intuitive and intelligible user interfaces, as TBR heavily relies on the user interacting with the MTMS.

The meta-model presented in this chapter, as well as the notion of Trace-Based Reasoning (TBR), are underlying a number of the works presented in the following chapters. Those will further demonstrate how this meta-model supports the user-centric co-construction of knowledge, taking into account the various contexts of use of that knowledge, and allowing multiple interpretations to coexist.

Notes

[1] An exception is a case that the computer can not handle, such as a division by zero or an access to a non-existing memory address. When an exception is raised by a program, that program is suspended and an exception handler is started, provided with the context in which the exception occurred. The term “error” is sometimes used instead of “exception”, but the very notion of error relates to an interpretation, even an appraisal.

[2] For example, services such as Google Analytics offer tools to precisely analyze the visits on a web site: http://www.google.com/intl/en/analytics/

[3] This assessment is based on legacy data, such as log files. Recent efforts proposing generic trace formats (Moreau and Missier 2013; Snell and Prodromou 2016) build on semantic web technologies and linked data principles, and are much more similar to our proposal.

[4] In this respect, let us point out how misleading the term “data” can be. It originally means “given”, which gives it an aura of neutrality or objectivity. In fact the data we get are not so much given as they are *taken* (observed, measured, extracted, captured, selected...) and therefore never independent of the processes set up to obtain them.

[5] Note that time-stamps in most operating systems are represented that way: as a number of time units (usually the second) since a give origin or “epoch” (typically 1970-01-01 on UNIX systems).

[6] There is another consequence: while it might seem trivial to convert time-stamps from a fine-grained unit to a coarser-grained unit, it is actually not always possible when using an opaque origin. For example, converting days to months can not be done accurately if we don’t have an absolute origin, as we do not know after how many days to change month. Less obviously, converting from hours (or minutes, or seconds) to days can not be done either: because of Daylight Saving Time, some days have 23 hours, and some have 25.

[7] Most collectors will be dedicated to one specific trace-model, with the constraints of that model hard-coded in them. However, one could imagine more generic collectors, able to inspect richly described trace-models in order to comply to them dynamically. Another perspective would be collectors able to dynamically edit trace-models, whenever they encounter a situation that the model can not represent.

[8] An efficient anonymization would actually require a more complex processing, as also the message bodies and attachments may contain information allowing to identify the persons involved. Still, those complex processes would still qualify as a rewriting transformation, according to our definition.

[9] The rationale of this choice is of course not to suggest that kTBS, or any MTMS, should in general be offshored to an external providers. Traces typically contain privacy-sensible information, and should obviously be kept in trustworthy locations. In a typical setup, the kTBS service would be deployed on the *same* server as the application.

Chapter bibliography

Aamodt, Agnar, and Enric Plaza. 1994. “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.” *AI Communications* 7 (1): 39–59.

Amer-Yahia, Sihem, Noha Ibrahim, Christiane Kamdem Kengne, Federico Ulliana, and Marie-Christine Rousset. 2014. “SOCLE: Towards a Framework for Data Preparation in Social Applications.” *Ingénierie Des Systèmes d’Information* 19 (3): 49–72. https://doi.org/10.3166/isi.19.3.49-72.

Bachimont, B. 2004. “Arts et Sciences Du Numérique : Ingénierie Des Connaissances et Critique de La Raison Computationnelle.” HDR, Université de Technologie de Compiègne.

Barazzutti, Pierre-Loup, Amélie Cordier, and Béatrice Fuchs. 2015. “Transmute: An Interactive Tool for Assisting Knowledge Discovery in Interaction Traces.” Research Report. Universite Claude Bernard Lyon 1 ; Universite Jean Moulin Lyon 3. https://hal.archives-ouvertes.fr/hal-01172013.

Besnaci, Mohamed, Nathalie Guin, and Pierre-Antoine Champin. 2015. “Acquisition de Connaissances Pour Importer Des Traces Existantes Dans Un Système de Gestion de Bases de Traces.” In *IC2015*. Rennes, France: AFIA. https://hal.archives-

ouvertes.fr/hal-01164384.

Champin, Pierre-Antoine, Alain Mille, and Yannick Prié. 2013. “Vers des traces numériques comme objets informatiques de premier niveau : une approche par les traces modélisées.” *Intellectica*, no. 59 (June): 171–204.

Cook, Jonathan E., and Alexander L. Wolf. 1998. “Discovering Models of Software Processes from Event-Based Data.” *ACM Transactions on Software Engineering and Methodology (TOSEM)* 7 (3): 215–249.

Cordier, Amélie, Marie Lefevre, Pierre-Antoine Champin, Olivier Georgeon, and Alain Mille. 2013. “Trace-Based Reasoning — Modeling Interaction Traces for Reasoning on Experiences.” In *The 26th International FLAIRS Conference*. http://liris.cnrs.fr/publis/?id=5955.

Deransart, Pierre, Mireille Ducassé, and Ludovic Langevine. 2002. “A Generic Trace Model for Finite Domain.” In *User-Interaction in Constraint Satisfaction*, edited by Barry O’Sullivan and Eugene C. Freuder. New York, NY, USA.

Estrin, Deborah. 2014. “Small Data, Where n = Me.” *Commun. ACM* 57 (4): 32–34. https://doi.org/10.1145/2580944.

Fielding, Roy Thomas. 2000. “Architectural Styles and the Design of Network-Based Software Architectures.” Doctoral dissertation, University of California, Irvine. http://www.ics.uci.edu/%7Efielding/pubs/dissertation/top.htm.

Fisher, Carolanne, and Penelope Sanderson. 1996. “Exploratory Sequential Data Analysis: Exploring Continuous Observational Data.” *Interactions* 3 (2): 25–34.

George, Sébastien, Christine Michel, and Magali Ollagnier-Beldame. 2013. “Usages Réflexifs Des Traces Dans Les Environnements Informatiques Pour l’apprentissage Humain.” *Intellectica*, no. 59: 205–241.

Harris, Steve, and Andy Seaborne. 2013. “SPARQL 1.1 Query Language.” W3C Recommendation. W3C. http://www.w3.org/TR/sparql11-query/.

Hitzler, Pascal, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. 2009. “OWL 2 Web Ontology Language Primer.” W3C Recommendation. W3C. http://www.w3.org/TR/owl2-primer/.

Hsieh, Cheng-Kang, Hongsuda Tangmunarunkit, Faisal Alquaddoomi, John Jenkins, Jinha Kang, Cameron Ketcham, Brent Longstaff, et al. 2013. “Lifestreams: A Modular Sense-Making Toolset for Identifying Important Patterns from Everyday Life.” In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, 5:1–5:13. SenSys ’13. New York, NY, USA: ACM. https://doi.org/10.1145/2517351.2517368.

Kaptelinin, Victor. 1996. “Computer-Mediated Activity: Functional Organs in Social and Developmental Contexts.” *Context and Consciousness: Activity Theory and Human-Computer Interaction*, 45–68.

Klyne, Graham, and Chris Newman. 2002. “Date and Time on the Internet: Timestamps.” RFC 3339. IETF. https://tools.ietf.org/html/rfc3339.

Kong Win Chang, Bryan, Marie Lefevre, Nathalie Guin, and Pierre-Antoine Champin. 2015. “SPARE-LNC : Un Langage Naturel Contrôlé Pour l’interrogation de Traces d’interactions Stockées Dans Une Base RDF.” In *IC2015*. Rennes, France: AFIA. https://hal.archives-ouvertes.fr/hal-01164383.

Laflaquiere, Julien, Lotfi S. Settouti, Yannick Prié, and Alain Mille. 2006. “Trace-Based Framework for Experience Management and Engineering.” In *Knowledge-Based Intelligent Information and Engineering Systems*, 1171–1178. Springer. http://link.springer.com/chapter/10.1007/11892960_141.

Marty, Jean-Charles, Thibault Carron, and Jean-Mathias Heraud. 2009. “Traces and Indicators: Fundamentals for Regulating Learning Activities.” In *Teachers and Teaching: Strategies, Innovations and Problem Solving*, 323–349. Nova Science. http://brainsharepubliconlinelibrary.tk/education/Teachers%20and%20Teaching%20Strategies.pdf#page=325.

Mathern, Benoît, Alain Mille, Amélie Cordier, Damien Cram, and Raafat Zarka. 2012. “Towards a Knowledge-Intensive and Interactive Knowledge Discovery Cycle.” In *20th ICCBR Workshop Proceedings*, edited by Juan A. Recio-Garcia Luc Lamontagne, 151–62. http://liris.cnrs.fr/publis/?id=5916.

Moreau, Luc, and Paolo Missier. 2013. “PROV-DM: The PROV Data Model.” W3C Recommendation. W3C. http://www.w3.org/TR/prov-dm/.

Ollagnier-Beldame, Magali. 2011. “The Use of Digital Traces: A Promising Basis for the Design of Adapted Information Systems?” *International Journal on Computer Science and Information Systems. Special Issue ”Users and Information Systems”* 6 (2): 24–45.

Sachan, Mrinmaya, Danish Contractor, Tanveer A. Faruquie, and L. Venkata Subramaniam. 2012. “Using Content and Interactions for Discovering Communities in Social Networks.” In *21st International Conference on World Wide Web*, 331–340. Lyon, France: ACM. http://dl.acm.org/citation.cfm?id=2187882.

Schreiber, Guss, and Yves Raimond. 2014. “RDF 1.1 Primer.” W3C Working Group Note. W3C. http://www.w3.org/TR/rdf-primer/.

Snell, James, and Evan Prodromou. 2016. “Activity Streams 2.0.” W3C Candidate Recommendation. W3C. http://www.w3.org/TR/activitystreams-core/.

Song, Minseok, Christian W. Günther, and Wil MP Van der Aalst. 2009. “Trace Clustering in Process Mining.” In *Business Process Management Workshops*, 109–120. Springer. http://link.springer.com/chapter/10.1007/978-3-642-00328-8_11.

Sporny, Manu, Gregg Kellogg, and Markus Lanthaler. 2014. “JSON-LD 1.0 – A JSON-Based Serialization for Linked Data.” W3C Recommendation. W3C. http://www.w3.org/TR/json-ld-syntax/.

Terrat, Helene. 2015. “Apports et Limites Des TICE Dans Les Apprentissages de La Langue Chez Les Élèves Handicapés

Moteurs Présentant Des Troubles Associés : Utilisation Des Traces Numériques Pour Favoriser l'apprentissage de La Langue Écrite." Phd Thesis, Université Lumière Lyon 2. http://www.theses.fr/s62275.

Van der Aalst, Wil MP, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and Anton JMM Weijters. 2003. "Workflow Mining: A Survey of Issues and Approaches." *Data & Knowledge Engineering* 47 (2): 237–267.

3. Trace based user assistance

This chapter focuses on the use of digital traces such as the ones formalized in the previous chapter, with an emphasis on uses that aim at assisting users in their task. I will present on the works described in three papers (Yahiaoui et al. 2011; Champin et al. 2010; Ginon et al. 2014). While they address the question in different fields and with various degrees of automation, they all demonstrate how traces can support the interpretation (and re-interpretation) of the user's activity.

3.1. Trace based redocumentation

In the work of Yahiaoui et al. (2011), the user's task is the *documentation* of a given activity, *i.e.* the production of a document reporting on that activity. The user may or may not be the same person who performed that activity (for example, a teacher can report on their student's activity). The target audience of the document might be the user themselves (in order to keep track of their analysis of the activity), a limited community (*e.g.* someone can document their use of an application to serve as a tutorial for colleagues or peers), or it can be publicly published.

In the case of computer-mediated activity, digital traces are usually available and can be considered as a form of documentation, although at a very low level of abstraction. We therefore proposed a redocumentation process illustrated in Fig. 3.1, and a tool named ActRedoc supporting that process. The available trace of the activity is assumed to comply with our meta-model (see Section 2.3) and is modeled in RDF. The "automated redocumentation" phase produces a transcription in English of all the information in trace, using NaturalOWL (Ganalis and Androutsopoulos 2007). While this new document is easier to read that the raw data from the m-trace, it is still verbose and mostly factual. Through the "interactive redocumentation" phase, the user can manually improve the produced document, removing or rephrasing existing segments, and adding new ones to enrich the document with analysis and opinions. This phase is repeated iteratively until the user is satisfied with the state of the document, and exports it to a standard format. Fig. 3.2 shows a screenshot of ActRedoc, displaying (from right to left) the available operations on the document, the individual segments constituting the structure of the document, and a preview of the resulting document.

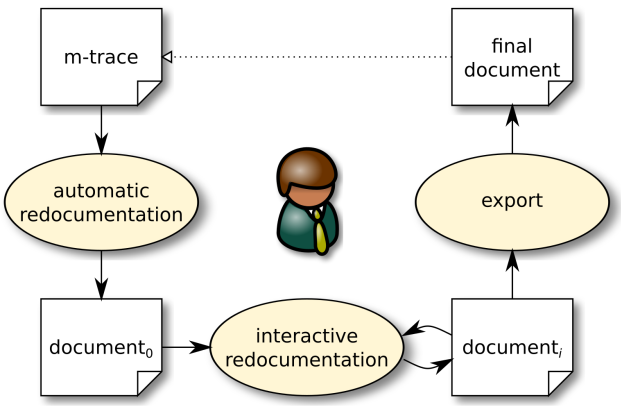
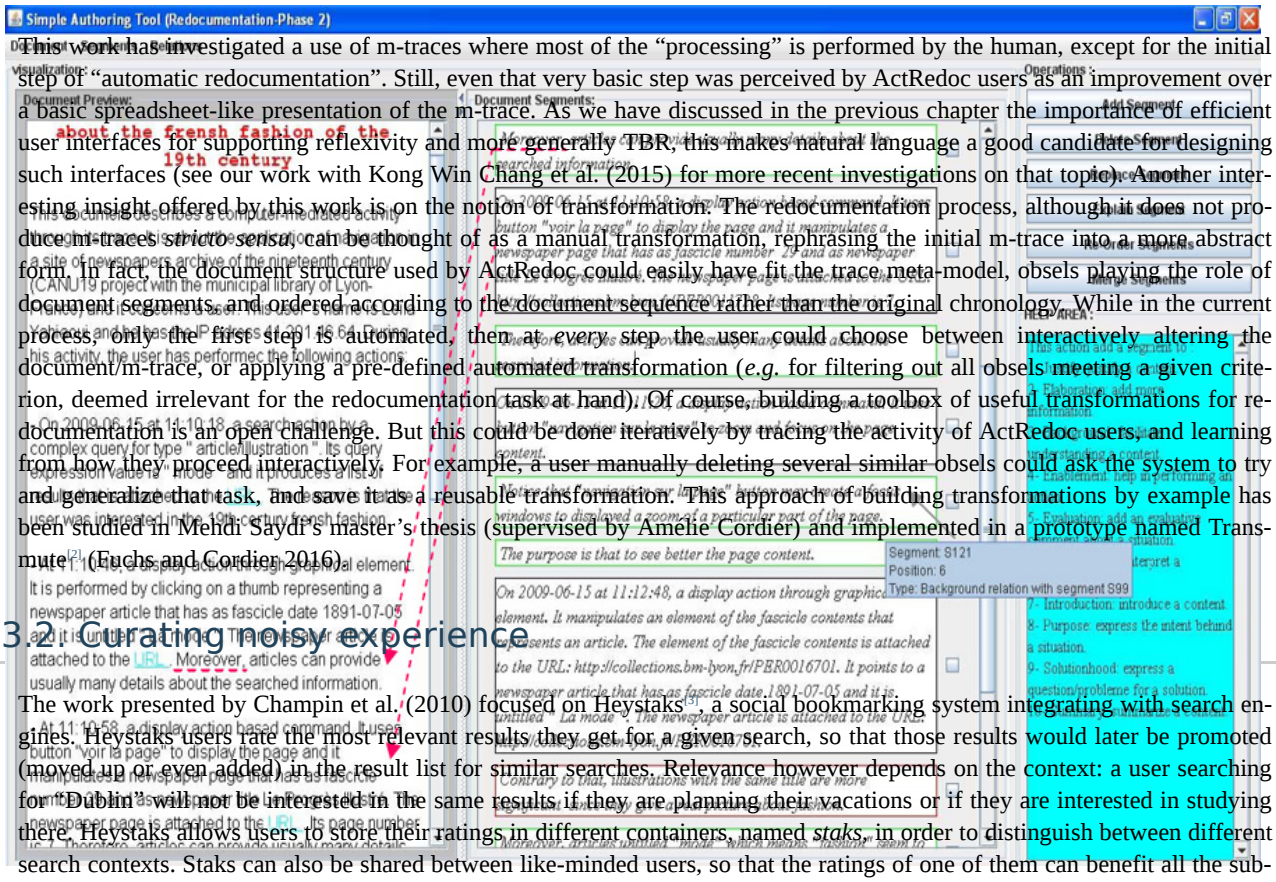


Fig. 3.1 The redocumentation process (Yahiaoui et al. 2011)

The underlying structure of the document is strongly inspired by the Rhetorical Structure Theory (RST) by Taboada and Mann (2006), which describes the different kinds of relations that can link documents segments. The benefit of RST is twofold. First, it provides users with guidelines in the redocumentation process, by eliciting the roles of the new segments they may want to add, and encouraging them to create a *narrative* above the factual sequence of events represented in the m-trace[1]. Second, those structural relations are kept in the final (HTML) form of the document, using RDFa (Adida and Birbeck 2008) for embedding semantic data, and the OntoReST ontology (Naja-Jazzar et al. 2009) to represent RST relations. Note that RDFa also allows us to link the segments of the final document to the related elements of the m-trace (which is represented by the dotted arrow on Fig. 3.1). The final document is therefore an improvement over the initial trace, not only for human users but also for machines (a perfect example of *co-construction* of knowledge as described in Section 2.4). Indeed, the rhetorical structure creates relations between obsels that could not be inferred without help from the user, all the more that it represents only one of many possible subjective interpretations of the activity (and so different users, or the same user in different contexts, would end up with different final documents).



This work has investigated a use of m-traces where most of the "processing" is performed by the human, except for the initial step of "automatic redocumentation". Still, even that very basic step was perceived by ActRedoc users as an improvement over a basic spreadsheet-like presentation of the m-trace. As we have discussed in the previous chapter the importance of efficient user interfaces for supporting reflexivity and more generally TBR, this makes natural language a good candidate for designing such interfaces (see our work with Kong Win Chang et al. (2015) for more recent investigations on that topic). Another interesting insight offered by this work is on the notion of transformation. The redocumentation process, although it does not produce m-traces *stricto sensu*, can be thought of as a manual transformation, rephrasing the initial m-trace into a more abstract form. In fact, the document structure used by ActRedoc could easily have fit the trace meta-model, obsels playing the role of document segments, and ordered according to the document sequence rather than the original chronology. While in the current process, only the first step is automated, then at *every* step the user could choose between interactively altering the document/m-trace, or applying a pre-defined automated transformation (*e.g.* for filtering out all obsels meeting a given criterion, deemed irrelevant for the redocumentation task at hand). Of course, building a toolbox of useful transformations for redocumentation is an open challenge. But this could be done iteratively by tracing the activity of ActRedoc users, and learning from how they proceed interactively. For example, a user manually deleting several similar obsels could ask the system to try and generalize that task, and save it as a reusable transformation. This approach of building transformations by example has been studied in Mehdi Saïdi's master's thesis (supervised by Amélie Cordier) and implemented in a prototype named Transmute[2] (Fuchs and Cordier 2016).

3.2 Curating noisy experience

The work presented by Champin et al. (2010) focused on Heystaks[3], a social bookmarking system integrating with search engines. Heystaks users rate the most relevant results they get for a given search, so that those results would later be promoted (moved up or even added) in the result list for similar searches. Relevance however depends on the context: a user searching for "Dublin" will not be interested in the same results if they are planning their vacations or if they are interested in studying there. Heystaks allows users to store their ratings in different containers, named *staks*, in order to distinguish between different search contexts. Staks can also be shared between like-minded users, so that the ratings of one of them can benefit all the subscribers of a given stak.

Heystaks is designed to accommodate user's practices as smoothly as possible; this is why it has been integrated in the result pages of main search engines, rather than providing a separate search interface. For the same reason, Heystaks does not force users to explicitly rate search results (although such an explicit rating is possible), but will also consider that visiting a link from the result page is a positive assessment of that result (although not as strong as an explicit rating). While this implicit rating is important to quickly populate staks, and hence perform better recommendations, it may cause a problem when users forget to select the appropriate stak when searching the web. In that case, the results visited by the user are wrongly recorded as relevant to the current stak, which will have a negative impact on future recommendations made by this stak.

Preventing this kind of mistake was not a solution, as there are more subtle ways irrelevant pages can be added to a stak: people may change their mind, and a page that did seem relevant once could be deemed irrelevant later. This is all the more true in social applications, where stak users may have divergent opinions. Instead, we wanted to provide stak owners with a tool for *curating* staks, by removing irrelevant ratings and possibly transferring them to a more appropriate stak, in order to fix mistakes and arbitrate disagreements, thus keeping the staks consistent and their recommendations efficient. To achieve this, we have proposed a meta-recommender system[4], recommending the three best staks for a every rated page. The main challenge was to build this system with the data we had, which already contained the kind of noise that we wanted to eliminate; in particular, we had no gold standard against which to evaluate our proposal.

We have first proposed a *popularity* measure to try and predict the relevance of a page to a stak. Intuitively, the popularity of a page p in a stak s increases with (1) the number of times p was rated in s, and (2) the frequency in s of every term used to search p (*i.e.* the number of *other* pages in s described with the same term). A preliminary user study has shown that pages with a high popularity are very likely to be relevant, while nothing can be told of pages with a low popularity (roughly 50% of them are relevant). So this popularity measure could be used to weight the evaluated accuracy of our meta-recommender system: "correct" recommendations (*i.e.* in accordance with the training data) should be sought for popular pages, but not necessarily for unpopular ones (as the training data is more likely to be wrong).

A natural idea was therefore to weight training instances based on their popularity, as we trust popular pages more. Surprisingly, classifiers trained that way did not improve significantly on classifiers trained with no popularity-weighting. This suggests that the relevance of popular pages could be learned from the raw data, which in a sense was encouraging. Furthermore, we noticed that weighted accuracy is better for popular pages, and that it does not vary linearly, as described in Fig. 3.3. It suggests that there are phases in the popularity spectrum, and that the upper third of that spectrum represents a stable subset of relevant pages. We call this subset the *stak kernel*. Pursuing our idea that popularity could be used to reduce the noise in training data, and thus improving meta-recommendation, we have trained classifiers with only pages from the kernel, and have shown

that it improves the accuracy of the Naive Bayes classifiers.

The standard way to build and evaluate a recommender systems is to use a gold standard, *i.e.* a set of data known to be exact, or at least reasonably good. It is very often built by having a small set of experts annotate the data. a long and costly process. That can be somehow alleviated by resorting on crowdsourcing platforms such as Amazon Mechanical Turk[5]. Still, in Web applications such as social bookmarking, it remains very hard to build a gold standard that covers the diversity of the actual data. The work presented in this subsection demonstrates the feasibility of an alternative approach: using the full amount of available data, even if it is known to be noisy. While the presented study is quite specific to Heystaks, we are confident the lessons learned and the proposed notion of kernel could apply to other contexts.

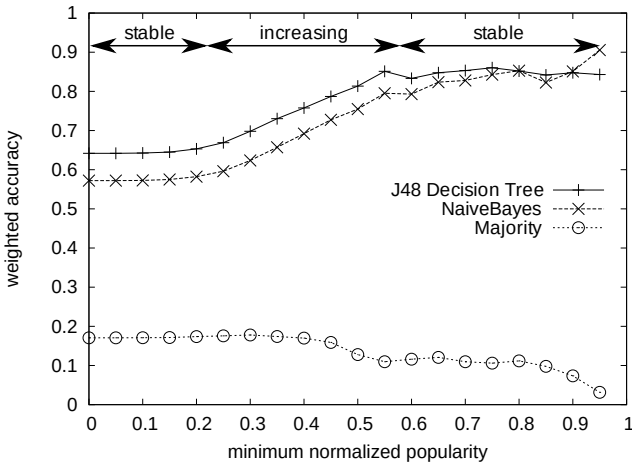


Fig. 3.3 The accuracy of J48 and Naive Bayes increases non linearly (Champin et al. 2010).

This plot shows how the weighted accuracy of our classifiers (trained with non-weighted instances) varies when classifying only pages above a certain popularity. This is not a bias in the data, as the majority classifier does not share this property.

Although this work did not rely on our meta-model, it is based on the same premises that digital traces (in the broadest sense) can be harnessed to produce knowledge and help assist users in their task. The traces available in Heystaks are however very synthetic, every page being described by a term vector, a compression of every individual rating of that page. This can be seen as a transformed trace of a possible primary trace where each rating would be stored as an obsel. Should we have access to that primary trace, we could examine alternative perspectives on staks. We could obviously filter out old ratings (or lower their weight) in order to forget all pages that were once relevant but are not anymore. But maybe more subtle temporal patterns could be used instead, that would represent more accurately opinion drift: for example, some pages are used only once in a while but remain relevant over time, while others “buzz” very intensely for a while, and are quickly forgotten. More generally, that primary trace would give access to the past evolution of a stak, possibly allowing to measure its stability (in terms of topic) or agreement (among its different users), some potentially useful indicators for the stak owners. Furthermore, that primary trace would provide the ability to trace back a page (or even an individual rating) to the user(s) who contributed it. This could help detect sub-communities in the users of a stak, a particular case being a malicious users trying to promote irrelevant pages for their own benefit.

3.3. Pro-active user assistance

In her PhD, Blandine Ginon (2014) proposed a generic architecture for setting up *epiphytic* assistance systems, and implemented it in a suite of tools called SEPIA. A biological metaphor, “epiphytic” means that such systems are added to a target application, without requiring any alteration of this application. Epiphytic assistance systems are motivated by the observation that, very often, the users of an application require assistance that the application itself does not provide. This can be due to their lack of practice, to the application being inherently complex, or simply to a bad design of its user interface. Still, modifying the application to include that assistance is not always feasible or practical.

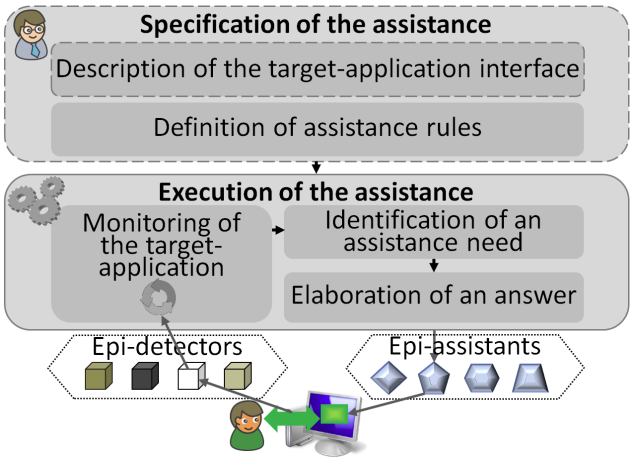


Fig. 3.4 The generic architecture implemented in SEPIA (Ginon et al. 2014)

In the first step of our approach (upper part of Fig. 3.4), the assistance system is first defined by the assistance designer, an expert user of the application, who needs not have access to the application source code, nor any advanced programming skill. Their task is mostly to state a number of rules of the form “whenever X happens, provide assistance Y”.

In the second step (lower part of Fig. 3.4), epiphytic detectors (or epi-detectors) are monitoring the user’s interactions with the target application, in order to trigger any assistance rule matching those interactions. Since rules can be more or less specific in describing the assistance to be provided, the assistance effectively enacted can be adapted and personalized, based on contex-

tual information (state of the application, user profile, traces of previous interactions).

As observing the user's activity is key to our approach, we have built a number of epi-detectors able to observe a large class of applications (Ginon, Champin, and Jean-Daubias 2013). Those epi-detectors gain access to the targer application thanks to accessibility APIs[6]. They communicate with the SEPIA assistance engine in the same protocol as used by kTBS, our reference MTMS implementation, which makes them reusable to build other trace-based applications. In fact, SEPIA uses kTBS to store the traces collected by the epi-detectors, as past interactions are one of the many parameters of personalization.

In order for assistance designers to easily define assistance rules, we have proposed aLDEAS (Ginon et al. 2014), a graphical language for expressing those rules. Fig. 3.5 shows two examples of rules described in aLDEAS (it is merely provided as an illustration here, the reader interested in a full description of the language should refer to the original paper). In addition to the language itself, we have provided a number of patterns, guiding the assistance designer in the definition of rules for the most common types of assistance. For example, we have a pattern for "guided tours" (a common assistance typically launched on the first execution of an application) and for step-by-step wizards (to assist the user in a complex task).

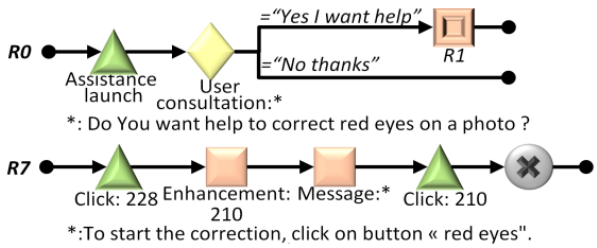


Fig. 3.5 Two assistance rules illustrating the use of aLDEAS (Ginon et al. 2014)

- Green triangles represent events (observed by epi-detectors).
- Yellow diamonds represent active information seeking by the assistance engine.
- Orange squares represent assistance actions.

We have performed a number of experiments and user studies in order to assess the good properties of SEPIA and its generic architecture.

First the aLDEAS language is expressive enough to represent a large variety of assistance systems encountered in existing applications. It fails to represent a few specific assistance systems, such as recommender systems (as used in e-commerce or social applications). This can be seen as an inevitable drawback of our epiphytic approach, as those systems are intimately linked to the data handled by the application. We also had potential assistance designers use aLDEAS to specify assistance systems of their own, and they have been satisfied with its expressive power (Ginon et al. 2014).

We have also shown that assistance systems operated by SEPIA are generally well accepted by users, and that they are actually helpful. In particular, we conducted an experiment in the context of an advanced programming course (Ginon, Champin, and Jean-Daubias 2013). All students had a background in programming, but some of them were not familiar with the Netbeans IDE[7], used in this course. SEPIA was used to assist them getting their bearings, leaving the teacher more time to answer questions related to the content of the course. After the course, the students answered a questionnaire, demonstrating an overall satisfaction with the assistance. Furthermore, the students passed a short test before and after the experiment, which showed that the assistance did indeed improve their knowledge of Netbeans. The teachers did also appreciate the assistance provided to their students, and express an interest in using the assistance again in the following semesters.

SEPIA provides a nice example of effective TBR (as presented in Section 2.4): with aLDEAS, assistance designers elaborate episode signatures that are further retrieved in real-time by the assistance engine, in order to trigger the epi-assistants. Furthermore, its epiphytic design allows several assistance systems to be defined for the same application. Each of them can address different kinds of users, different kinds of tasks, or simply provide a different perspectives of how to use the application.

SEPIA's main limitation with respect to TBR is that it does not use transformations: the only events currently taken into account in assistance rules are those of the primary trace, the raw events collected by the epi-detectors (*e.g.* mouse click on interface components, individual keystrokes). This limitation could easily be lifted, though, as aLDEAS is generic enough to support more abstract kinds of events, which could be sought in transformed traces rather than directly from the epi-detectors. In fact, aLDEAS itself could be used to specify *transformation rules*[8]; while assistance rules result in assistance actions, transformation rules would result in the creation of new (more abstract) obsels (*e.g.* copy-paste, send an e-mail to a colleague) which could in turn participate in triggering other rules. This would make the task of assistance designers more modular, decoupling the interpretation of low-level activity from the decision of performing an assistance action.

The works presented above demonstrate how interaction traces (in various forms) can be tapped as knowledge about the user's activity. They also emphasize the value of combining deterministic automated processings with subjective human interpretations, in a virtuous cycle that reveals the richness of those traces.

Notes

[1] Similar approaches have been use by Kim et al. (2002) and Mulholland, Wolff, and Collins (2012) in the context of digital heritage.

[2] http://tbs-platform.org/tbs/doku.php?id=tools:transmute

[3] http://heystaks.com/. Note that Heystaks has evolved a lot since this work was published, and that description may not match exactly the current state of the system.

[4] It is a *meta*-recommender system in the sense that it recommends a stak, which is itself a recommender system (Schafer, Konstan, and Riedl 2002).

[5] https://www.mturk.com/

[6] Those APIs are originally aimed at people with disabilities. They allow tools such as screen readers or Braille displays to access (and even interact with) other applications. Note that such tools are epiphytic systems, according to our definition above, hence our interest for accessibility technologies. Nevertheless, those APIs are not largely standardized (they vary with operating systems, graphical toolkits, *etc*.), so we had to use several of them to cover a wide range of applications.

[7] https://netbeans.org/

[8] Champalle et al. (2011) have already proposed to express a transformation as a set of rules, each rule being responsible of producing obsels of a given type. In that work, though, rules were expressed in SPARQL (Harris and Seaborne 2013).

Chapter bibliography

Adida, Ben, and Mark Birbeck. 2008. “RDFa Primer.” W3C Working Group Note. W3C. http://www.w3.org/TR/xhtml-rdfa-primer/.

Champalle, O., K. Sehaba, A. Mille, and Y. Prie. 2011. “A Framework for Observation and Analysis of Learners’ Behavior in a Full-Scope Simulator of a Nuclear Power Plant - Approach Based on Modelled Traces.” In *2011 11th IEEE International Conference on Advanced Learning Technologies (ICALT)*, 30–31. https://doi.org/10.1109/ICALT.2011.17.

Champin, Pierre-Antoine, Peter Briggs, Maurice Coyle, and Barry Smyth. 2010. “Coping with Noisy Search Experiences.” *Knowledge-Based Systems*, Artificial Intelligence 2009AI-2009The 29th SGAI International Conference on Artificial Intelligence, 23 (4): 287–94. https://doi.org/10.1016/j.knosys.2009.11.011.

Fuchs, Béatrice, and Amélie Cordier. 2016. “Interprétation Interactive de Connaissances à Partir de Traces.” In *Ingénierie Des Connaissances 2016*, edited by Nathalie Pernelle, 167. Actes Des 27es Journees Francophones d’Ingenierie Des Connaissances - IC 2016. Montpellier, France: Sandra Bringay. https://hal.archives-ouvertes.fr/hal-01343518.

Ganalis, Dimitrios, and Ion Androutsopoulos. 2007. “Generating Multilingual Descriptions from Linguistically Annotated OWL Ontologies: The NaturalOWL System.” In *11th European Workshop on Natural Language Generation (ENLG 2007)*, 1–4. Schloss Dagstuhl, Germany.

Ginon, Blandine. 2014. “Modèles et outils génériques pour mettre en place des systèmes d’assistance épiphytes.” Thèse de Doctorat en Informatique, INSA de Lyon, Université de Lyon. http://liris.cnrs.fr/publis/?id=6885.

Ginon, Blandine, Pierre-Antoine Champin, and Stéphanie Jean-Daubias. 2013. “Collecting Fine-Grained Use Traces in Any Application without Modifying It.” In *Workshop EXPPORT from the Conference ICCBR*. http://liris.cnrs.fr/publis/?id=6141.

Ginon, Blandine, Stéphanie Jean-Daubias, Pierre-Antoine Champin, and Marie Lefevre. 2014. “ALDEAS: A Language to Define Epiphytic Assistance Systems.” In *Knowledge Engineering and Knowledge Management (EKAW’14)*, edited by Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, 8876:153–64. LNCS. Linköping, Sweden: Springer. https://doi.org/10.1007/978-3-319-13704-9_12.

Harris, Steve, and Andy Seaborne. 2013. “SPARQL 1.1 Query Language.” W3C Recommendation. W3C. http://www.w3.org/TR/sparql11-query/.

Kim, Sanghee, Harith Alani, Wendy Hall, Paul Lewis, David Millard, Nigel Shadbolt, and Mark Weal. 2002. “Artequakt: Generating Tailored Biographies from Automatically Annotated Fragments from the Web.” http://eprints.ecs.soton.ac.uk/6913.

Kong Win Chang, Bryan, Marie Lefevre, Nathalie Guin, and Pierre-Antoine Champin. 2015. “SPARE-LNC : Un Langage Naturel Contrôlé Pour l’interrogation de Traces d’interactions Stockées Dans Une Base RDF.” In *IC2015*. Rennes, France: AFIA. https://hal.archives-ouvertes.fr/hal-01164383.

Mulholland, Paul, Annika Wolff, and Trevor Collins. 2012. “Curate and Storyspace: An Ontology and Web-Based Environment for Describing Curatorial Narratives.” In *The Semantic Web: Research and Applications (ESWC’12)*, edited by Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, 7295:748–62. LNCS. Springer. https://doi.org/10.1007/978-3-642-30284-8_57.

Naja-Jazzar, Hala, Nishadi de Silva, Hala Skaf-Molli, Charbel Rahhal, and Pascal Molli. 2009. “OntoReST: A RST-Based Ontology for Enhancing Documents Content Quality in Collaborative Writing.” *INFOCOMP Journal of Computer Science* 8 (3): 1–10.

Schafer, J. B., J. A. Konstan, and J. Riedl. 2002. “Meta-Recommendation Systems: User-Controlled Integration of Diverse Recommendations.” In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, 43–51. ACM New York, NY, USA.

Taboada, Maite, and William C. Mann. 2006. “Rhetorical Structure Theory: Looking Back and Moving Ahead.” *Discourse Studies* 8 (3): 423–59. https://doi.org/10.1177/1461445606061881.

Yahiaoui, Leila, Yannick Prié, Zizette Boufaïda, and Pierre-Antoine Champin. 2011. “Redocumenting Computer Mediated Activity from Its Traces: A Model-Based Approach for Narrative Construction.” *Journal of Digital Information (JoDI)* 12 (3). https://journals.tdl.org/jodi/index.php/jodi/article/view/2088.

4. Web and Semantic Web

Since the Web was invented some 25 years ago, its pervasiveness has grown to the point of becoming trite. The Web has become a primary means of communication between colleagues, family members and friends; it is being used for work, leisure, shopping, paying taxes, finding a restaurant, a job, or a date... It connects our computers, our phones, our TV-sets, and a growing number of other (and sometimes unexpected) items, such as fridges, cars, electricity-meters or flowerpots.

Not only has it changed the amount of information available to us, it has changed dramatically how we acquire, handle and use this information, and turn it into knowledge. Therefore, it brings both a challenge and an opportunity, to better understand and to assist these new practices (Berners-Lee et al. 2006, chap.5). Since these are computer-mediated, using digital traces seems a natural way to achieve those goals. And since the Web is so pervasive, the available information is bound to be interpreted differently by different agents, or even by the same agent in different contexts, hence a need to take ambivalence into account.

This chapter gathers a variety of works that we have done in a research context, but also in the context of standardization groups, more precisely groups in the W3C, of which Université de Lyon is a member since 2012. Those works are presented along three dimensions: the use of activity traces on the Web, the acknowledgment of ambivalence in Web technologies and Web standards, and how those aspects may lead to new paradigms to design Web applications.

4.1. Putting Web traces to good use

As stated above, our activity on the web accounts so much for our lives that the traces of that activity can provide a huge amount of knowledge about us. Indeed Web companies such as Facebook (Kramer 2012) or Google (Bonneau et al. 2015) have a long history of tapping the traces of their users to provide better targetted services (and advertisements). Some of them, as Netflix[1] or Yahoo[2], have even opened some of their data to a wider research community, in order to find better ways to capture the collective knowledge of a large number of users.

However, as stated in Section 2.2, our approach is more focused on capturing the individual experience of each traced user. In her presentation of the “small data” approach, Estrin (2014) explains how health problems could be detected earlier through changes in individual behavioral patterns. While of primary importance for the users themselves, those changes may be less relevant for the companies currently holding and exploiting the users’ traces. Furthermore, many users would have concerns about those companies monitoring their health status.

Another effect of our traces being exploited out of our control is what Pariser (2011) calls the “filter bubble”: the fact that the content provided to us by search engines and social networks are tailored to meet our preferences, as computed from our activity traces. Of course it can be seen as a benefit, helping us find what we are looking for in an overwhelming amount of information. But on the other hand, those social tools paradoxically *isolate* us from whole parts of the Web, and this is all the more pernicious that they keep an aura of exhaustivity and objectivity (after all, *they* have access to the whole Web).

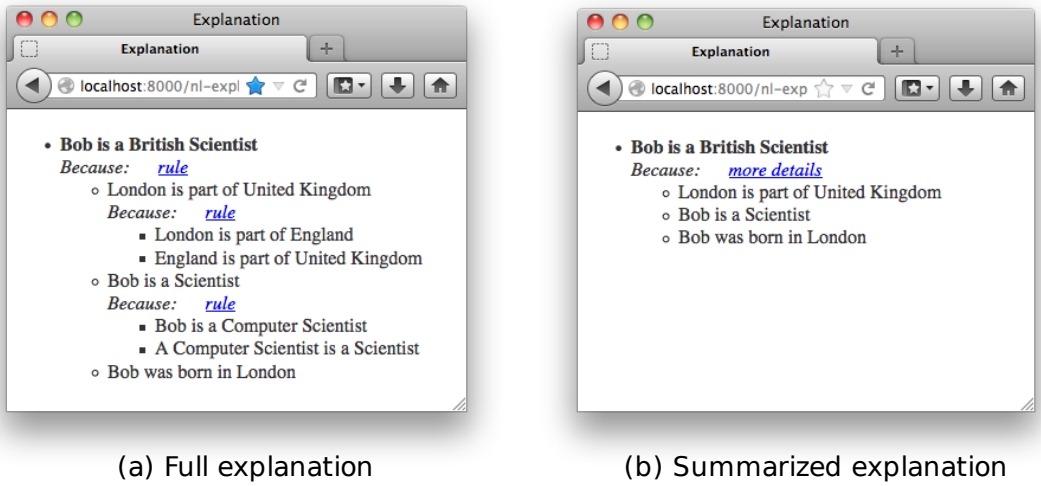
Querying the Web

Pariser advocates a way to disable the personalization mechanisms, in order to be able to access the Web more objectively. This has been supported by some search engines such as DuckDuckGo[3] or Qwant[4]. An alternative, and a way to compensate the lack of that feature in other systems, is to provide users with access to their traces and tools to analyze them. While this would not suppress the filter bubble, at least it would allow users to know what bubble they are in, and how their behavior alters that bubble.

The work about Heystaks presented in Chapter 2 provides such an alternative: by choosing a given stak as the context of their Web searches, users consciously select their filter bubble. Furthermore, they can register to as many staks as they like, allowing them to chose the bias on each of their search results[5]. Finally, the stak curation tools give users access to the full history

of searches performed in a given stak, providing insight on which parts of one's behavior participates to stak recommendations.

In the PhD work of Rakebul Hasan (2014b), we focused on analyzing and synthesizing the information of traces, to help users understand and predict the outcome of querying linked data on the Web. First, machine learning techniques have been used on a set of SPARQL query evaluations, in order to identify which features of a query are predictive of its execution time. This differs from other approaches, which rather rely on the structure of the queried data. Second, by tracing the execution of the query engine itself, explanations in the form of *why-provenance* are generated and provided to users, in order to help them understand the query results, especially when inferences are involved (see Fig. 4.1-a). Such explanations can in turn be published as linked data, using RQ4 (Hasan 2014a), an extension of the PROV-O vocabulary (Lebo, Satya Sahoo, and McGuinness 2013). Finally, as those explanations can become quite verbose for complex queries, a summarization process for explanation (illustrated in Fig. 4.1-b) has been proposed and evaluated.



(a) Full explanation (b) Summarized explanation

Fig. 4.1 Examples of a full explanation and a summarized explanation (Hasan 2014b)

Traces for learning on the Web

We have also proposed a number of innovative uses of Web traces in the context of COAT[6], an exploratory project aimed at studying the research opportunities of e-learning, more specifically of Massive Open Online Courses (MOOCs). In those systems, learners are so many and so heterogeneous that standard indicators and monitoring tools can not be used (Buffat, Mille, and Picasso 2015); more flexible and scalable ones must be proposed. Furthermore, with MOOCs, the learning activity is no more confined to the hosting platform, as the learners are often pointed to external contents. Meaningful indicators can therefore only be computed by monitoring the learners' activity inside *and outside* the MOOC platform.

TraceMe[7] is a browser extension designed to trace the whole browsing activity of a user. Since it runs on the client-side, it is not restricted to tracing the activity on a given server. In addition, it can trace interactions that would be otherwise invisible to the server (such as navigation through internal links inside a page, interactions with an embedded video or audio player, *etc.*). But most importantly, TraceMe has to be installed voluntarily by the user, who may enable or disable the tracing at any moment. TraceMe can also be configured with several MTMSs, and the user can choose on which of them the traces should be collected. For example, a user may collect her traces on the MTMS provided by the MOOC when she is browsing content in relation with the course, and on a personal MTMS when she is browsing for other purposes. This kind of practice is actually encouraged in the emerging standard Tin Can (also known as the Experience API[8]), in which the notion of Learning Record Store (LRS) closely resembles our notion of MTMS. We are currently working on making TraceMe and kTBS (our MTMS reference implementation) interoperable with Tin Can[9].

SamoTraceMe[10] is a Web application aimed as a companion application to TraceMe. As illustrated by Fig. 4.2, it provides various ways to visualize one's trace, as well as tools to customize those visualizations. In order to help learners and teachers to analyze traces, SamoTraceMe also provides tools to build, execute and share indicators, *i.e.* synthetic representation of the information conveyed by the trace. More precisely, the "Indicator Back-Office" provides user-friendly tools to transform traces (as described in Chapter 2), and query them using the natural-language interface proposed by Kong Win Chang et al. (2015). That way, users can explore and design new indicators, better suited to MOOCs than those available in the litterature, as emphasized above. For the same reason, SamoTraceMe not only encourages the building of new indicators, but also their sharing with others (in the last tab "Indicator Store"). In that sense, it is a tool for learners and teachers as much as for researchers in

education sciences, making all of them actors of that research.

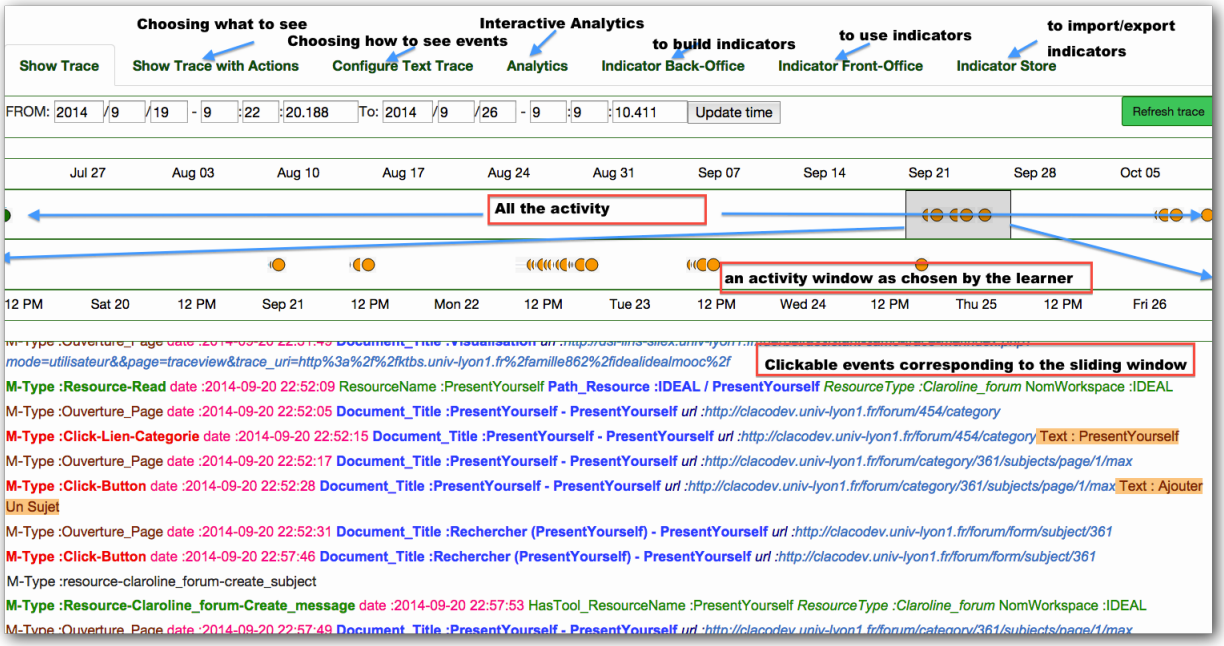


Fig. 4.2 The main screen of SamoTraceMe (Buffat, Mille, and Picasso 2015)

It contains (from top to bottom): tabs providing access to the various functionalities; a graphical timeline representing the whole trace; a graphical timeline zooming on the time-window selected in the above timeline; a hyper-textual representation of the selected time window, as a list of events (obsels).

Recently, we have started working on Taaabs[11], a set of reusable components for visualizing and interacting with traces, aimed at capitalizing on the experience acquired with SamoTraceMe and other works (Barazzutti, Cordier, and Fuchs 2015; Kong Win Chang et al. 2015). Taaabs relies on Web Components, a coming W3C standard (Glazkov 2016), so that each component is available as a custom HTML element. The goal is to make it as easy as possible for developers to add trace-based functionalities to their applications. A longer term goal is to allow end-users to interactively build their customized dashboard by drag-and-dropping visual components.

Interoperability

We also aim to improve the integration of our trace meta-model (as described in Chapter 2) with other models gaining momentum on the Web. One of them is PROV (Moreau and Missier 2013; Lebo, Satya Sahoo, and McGuinness 2013), a standard data-model for representing provenance information on the Web, hence concerned with *traceability*. A central element of this data-model (depicted in Fig. 4.3) is the notion of *Activity*, during which the object of the provenance information (the *Entity*) was produced or altered. This notion of Activity has an obvious kinship with our notion of obsel. PROV also defines interesting relations between entities. An entity *specializes* another entity “if it shares all aspects of the latter, and additionally presents more specific aspects”; for example, the second edition of a book is a specialization of that book (as a work); P-A. Champin as a researcher is a specialization of P-A. Champin as a person; P-A. Champin as mentioned in this document is a specialization of P-A. Champin as a researcher. While the specialized entity inherits the properties of the general one, the opposite is not true. This allows to make assertions with a limited scope, hence to have different interpretations coexist in the same knowledge base. PROV has its own data model, but defines a mapping with RDF.

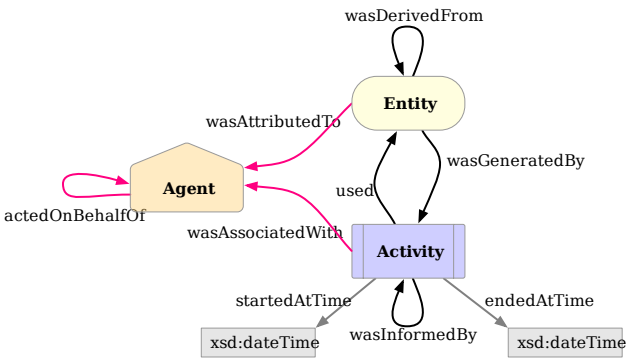


Fig. 4.3 PROV Core Structure (Lebo, Satya Sahoo, and McGuinness 2013)

Another model for representing traces on the Web is Activity Streams, an independent format (J. M. Snell et al. 2011) recently endorsed by the W3C Social Web Working Group (J. Snell and Prodromou 2016). This format is intended to represent actions

performed by a user, typically in the context of a social network application. It is extensible, and the most recent version is based on JSON-LD (Sporny, Kellogg, and Lanthaler 2014), making it interoperable with RDF and other Semantic Web technologies.

As mentioned above, Tin Can[8] is another format for capturing traces, focused on the domain of e-learning. Originally based on Activity Streams, it has then slightly diverged from it. In particular it is not based on the RDF data model, but De Nies et al. (2015) have proposed an approach to bridge this gap, by mapping it to PROV. Tin Can is currently being considered in the Hubble project[12] as an interoperability layer between the different platforms of the partners (including our own).

In his master's thesis, Cazenave-Lévêque (2016) has compared those emerging standards (and others) to our meta-model. While the latter focuses exclusively on time-anchored elements (obsels), the others allow to describe a number of objects that are not (or at least not explicitly) related to time. We have therefore proposed an extension of our meta-model, where additional information can be attached to a trace, or to an obsel. The first case is useful for representing background knowledge, that is assumed to hold for the whole duration of the trace (such as names and addresses of the persons involved in the activity). The second case is useful for representing contextual information captured at the same time as the obsel, and that is only assumed to hold for the duration of that obsel (such as the heart rate or mood of the person performing an observed action). We have shown that those extensions allow us to capture the semantics of PROV and Tin Can in our meta-model, and hence to integrate existing Web traces in an MTBS.

4.2. Ambivalence on the Web

Ambivalent documents

The distinction, in a document, between its physical structure and its logical structure, has long been identified and theorized. This is, in particular, why the original HTML (mixing concerns about both structures) was later split into CSS (addressing the physical structure, *i.e.* presentation) and the cleaner HTML 4 (restricted to the logical structure). But this dichotomy, however useful, is not always sufficient to capture the different overlapping structures of more complex documents, such as acrostics[13], multimedia or hypermedia documents. In such cases, the multiple structures can lead to multiple interpretations.

In 2003, a working group funded by the Rhône-Alpes region was formed in Lyon to investigate that topic. We proposed a formal model and an XML-based syntax for representing documents with an arbitrary number of structures (Abascal et al. 2003, 2004). This model allowed us to represent not only a multi-structured document, but also a curated corpus of such documents, where the corpus is considered itself as a document, with its own additional structures spanning the documents it contains (for example, a thematic index). It was also a way to capture altogether the original structures of a document, and the *annotations* added afterwards by a community of reader. This last point was further explored in the works described in the next chapter.

But the evolution of the Web has also forced ambivalence into HTML itself. The growing importance of the mobile Web, and the diversity of the devices used to display HTML content, have made *responsiveness* an unavoidable part of web design (Marcotte 2010). The goal is to design HTML contents such that they can be equally easy to read and use on devices with different screen sizes, but also with different interaction modalities (mouse and keyboard, touchscreen, remote control...). One could argue that, in responsive design, only presentation (managed by CSS) and interactions (managed by Javascript) are changed, but the logical structure conveyed by HTML is the same, and therefore there is no ambivalence here. However, responsive design can not be reduced to adding clever CSS and Javascript to any HTML document; the document structure has to be designed accordingly: unresponsive HTML is strongly coupled to one particular presentation, responsive HTML is abstract enough to be presented, hence *interpreted* [14], in various ways, so we argue that it has a form of ambivalence.

Weaving documents and data

HTML has also been used in various ways to convey other information that the document structures it was initially meant for. This provides another nice example of the need to support ambivalence in an open context as the Web, and how this has actually been achieved.

SHOE (Heflin, Hendler, and Luke 1999), the Simple HTML Ontology Extensions, is an early attempt at realizing the Semantic Web by embedding machine-readable knowledge into human-targetted HTML documents. It allows any Web page to define its own ontology (as a set concepts and relations) or reuse an existing one, then to describe its content using this ontology (see the example in Listing 4.1). The formal semantics of SHOE is purposefully lightweight, to ensure the scalability of the corresponding reasoning algorithms.

Let us note that, although SHOE uses HTML to "host" machine-readable data, this data is still quite separate from the human-readable content (it is enclosed in specialized HTML tags). Extending HTML in that case is mostly a way to ease the publication of ontologies, and to keep a link between human-readable and machine-readable information, although at a relatively

coarse granularity (that of the page).

RDFa (Adida and Birbeck 2008), on the other hand, proposes to use HTML attributes to represent fine-grained annotation of the human-readable content, which can in turn be interpreted as RDF data (see the example in Listing 4.2). This strongly reduces redundancy compared to SHOE (or other approaches where RDF is published in a completely separate file), making it more space-efficient and, more importantly, easier to author and maintain. Styles, Shabir, and Tennison (2009) have even proposed a system where, through editing an RDFa-annotated document with a WYSIWYG interface, users not only update the visible human-readable content, but also the underlying machine-readable data, even without being aware of the latter.

In parallel to RDFa, the Microformats community[15] has been pursuing a similar goal (fine-grain annotations of HTML content to provide machine-readable information) but without willing to commit to RDF for their data model. Later, Schema.org[16] was announced by a consortium of search engines (Goel and Gupta 2011), confirming the trend to use HTML to address both machines and humans, by weaving two complementary kinds of information.

Listing 4.1 SHOE example (from http://www.cs.umd.edu/projects/plus/SHOE/)

```
<P> Hi, this is my web page.
    I am a graduate student and a research assistant.
<P> Also, I'm 52 years old.
<P> My name is George Stephanopolous.

<INSTANCE KEY="http://www.cs.umd.edu/users/george/">

     <USE-ONTOLOGY
               ID="cs-dept-ontology"
               URL="http://www.cs.umd.edu/projects/plus/SHOE/onts/cs.html"
               VERSION="1.0"
               PREFIX="cs">

     <CATEGORY NAME="cs.GraduateStudent">
     <CATEGORY NAME="cs.ResearchAssistant">

     <RELATION NAME="cs.name">
            <ARG POS=TO VALUE="George Stephanopolous">
     </RELATION>
     <RELATION NAME="cs.age">
            <ARG POS=TO VALUE="52">
     </RELATION>
 </INSTANCE>
```

Listing 4.2 RDFa example (adapted from the SHOE example above)

```
<div prefix="cs: http://www.cs.umd.edu/projects/plus/SHOE/onts/cs.html#"
     about="http://www.cs.umd.edu/users/george/">
<p> Hi, this is my web page.
    I am a
      <span rel="rdf:type" resource="cs:GraduateStudent">
          graduate student</span>
    and a
      <span rel="rdf:type" resource="cs:ResearchAssistant">
          research assistant</span>.
<p> Also, I'm <span property="cs:age">52</span> years old.
<p> My name is <span property="cs:name">George Stephanopolous</span>.
</div>
```

Ambivalent data

Of course, data on the Web is not only embedded in HTML, but also available under many other formats, and working with heterogeneous data often requires to reinterpret them. For example, in order to ease the composition of heterogeneous data-providing services, we have proposed to interpret their output as the result of SPARQL queries (Barhamgi et al. 2007). More precisely, we considered all the available data as a virtual global RDF graph; then we attached to each service the parameterized SPARQL query on that graph that would return the same data as the service. Our system was able to solve a SPARQL query by decomposing it into sub-queries corresponding to the available services, effectively computing the appropriate ser-

vice composition.

Data heterogeneity is also a critical problem in the field of multimedia documents. While this obviously applies to the multimedia data itself, this causes even more acute problems with the *meta-data* describing the media object (title, author, date of creation, *etc.*). Indeed, meta-data formats are not differing in their syntax only, but also in their terminologies, the granularity of the represented information, and more generally, the underlying concepts.

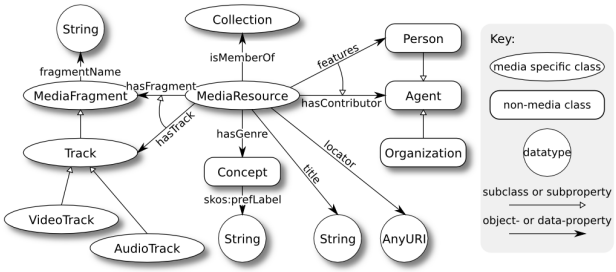


Fig. 4.4 Class model excerpt of the *Ontology for Media Resource 1.0* (Stegmaier et al. 2013)

The W3C Multimedia Annotation Working Group (MAWG) was created to address the problem of meta-data heterogeneity, among a list of widespread formats. We proposed an ontology (Lee et al. 2012) capturing the core meta-data concepts identified in those formats (illustrated in Fig. 4.4), as well as mappings from each format to this ontology. This mapping allows to re-interpret legacy meta-data into the common frame of reference provided by the MAWG ontology.

I was also a member of the RDF 1.1 Working Group, who was chartered in 2011 to update the 2004 version of that standard, based on feedback from the community. An important action of that group was to endorse JSON-LD (Sporny, Kellogg, and Lanthaler 2014) as a concrete syntax for RDF based on the popular format JSON. JSON-LD allows to attach a *context* to any JSON data, either in the data itself, or *via* out-of-band information. The role of a JSON-LD context is to disambiguate and elicit the semantics of the data, by allowing to convert it to an RDF dataset. Legacy JSON can therefore be kept as is, and still be processed by *ad hoc* programs ignoring the context; however it can also be processed by more generic programs, with no prior knowledge of the data structure, but able discover it and possibly make inferences with the corresponding RDF data. As such, JSON-LD is more than just another RDF serialization: it makes RDF data easier to consume for the average Web developer, and it makes legacy JSON data easily (and often quite directly) interoperable with the rest of the Web of data. In fact, JSON-LD is an ambivalent data format, designed to be equally suiting two communities with different expectations.

In another work (Steiner et al. 2014), we have proposed to apply a similar approach to another format, WebVTT (Pieters 2016). WebVTT is the proposed standard to represent text tracks for HTML5 videos. The most common use of those text tracks is to encode subtitles or captions, but they allow other uses, such as describing chapters or time-anchored other meta-data. To make that information usable outside Web browsers, we therefore suggest that WebVTT should and can be interpretable as Linked Data. We have proposed an RDF vocabulary capturing the concepts of WebVTT, and implemented a prototype converting any WebVTT into a standard RDF concrete syntax. This was made possible by leveraging Media Fragment URIs (Troncy et al. 2012), a standard way to identify fragments of a multimedia resource with a URI. A future development of this work would be map our vocabulary to the emerging Web Annotation Data Model (Sanderson, Ciccarese, and Young 2016).

4.3. Rethinking Web application design

The efforts described above to make the Web a machine-friendly data space, in addition to being a human-friendly document space, have mostly focused on helping machines read and understand those data. But the Web is not “read-only”, users are not merely consuming information. They are as well producing it, either directly through posting, commenting, rating (using HTML forms or Javascript applications) or indirectly in the interaction traces they leave on the sites they visit.

The Web Services community has striven to enable machines to interact with Web applications, in a way similar to what human users do. There is indeed an ever-growing number of Web APIs[17], but the semantics of each API has to be hard-coded into software clients, making them unable, unlike humans, to adapt to changes or discover and use similar services. So while the Web was initially designed as a unified and loosely coupled environment, where any client could connect to any server, the Web of services has become a siloed and strongly coupled environment, where an *ad hoc* client is required for every new service. Although this problem should be mitigated by following the REST architectural style (Fielding 2000), Baker (2008) notices that many self-proclaimeds RESTful services are lacking an important feature of that style: Hypermedia controls. Fielding defines Hypermedia as “the presence of application control information embedded within (...) the presentation of information”. In other terms, in order to interact with a service, a client should require almost no out-of-band static knowledge about that service, and mostly rely on the information dynamically retrieved from it.

In order to solve that problem, the W3C Linked Data Platform (LDP) working group was chartered to propose a standard RESTful way to interact with linked data. I was part of that group, who published the LDP recommendation (Speicher, Arwe, and Malhotra 2015). That recommendation specifies the operations that a compliant service must support, and how such a service can advertise to clients that it complies with LDP. It also defines a notion of *container*, allowing clients to dynamically

discover or create new resources. As such, LDP meets the Hypermedia requirements stated above: any LDP-aware client can readily interact with any LDP-compliant server, using only the information provided by that server. However, those interactions are somehow limited. For example, LDP provides no standard way for the server to declare the types of resources it may contain, what properties are allowed or required, *etc.*

That limitation is not a problem if LDP servers are only expected to passively store resource representations, leaving client applications to deal with specific resource semantics (*i.e.* application logic). This is the approach chosen by Mansour et al. (2016) for the SOLID platform. In this platform, applications do not host user's data on their own server (as is currently the case in most Web applications); instead, data storage is delegated to an LDP server owned and controlled by the user (see Fig. 4.5). Beyond the obvious benefit in terms of privacy, this provides users with more control over their data: they can chose which part of those data each application will be authorized to use. They can even chose to grant an application with access the data produced by another one, while currently, this choice is ultimately made by the application providers[18]. On the other hand, while applications can communicate asynchronously through the data they store in the personal LDP server, this approach leaves entirely open the question of a more direct composition of services.

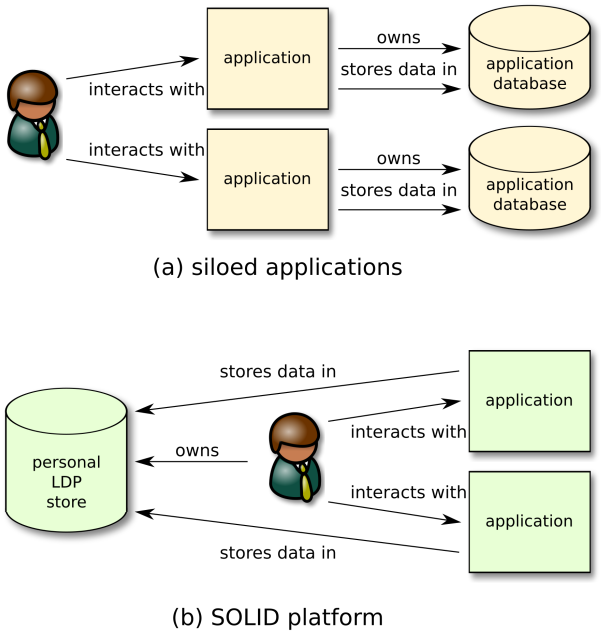


Fig. 4.5 The SOLID platform compared to the "silo" model

In parallel to LDP, the W3C community group Hydra has chosen to tackle that problem differently (Lanthaler and Gütl 2013). Like LDP, Hydra advocates the RDF data-model, by encouraging the use of JSON-LD. Unlike LDP, Hydra does not aim to define a unique interaction model with resources, but instead provides an RDF vocabulary for describing the particular Hypermedia controls of each service. More precisely, a Hydra service publishes an API documentation describing (1) the types of resources that compose the service, with their properties, and (2) which operations are available on different resources and property values, and how they can be performed in terms of HTTP requests. Hydra provides abstraction for the most common operations (resource creation, modification and deletion, search, pagination). More specific semantics can either be interpreted by the end-user (for interactive Hydra clients), built-in into specialized clients, or discovered dynamically using Linked Data principles (dereferencing unknown terms to retrieve their definition). One success story of Hydra is the Triple Pattern Fragment (TPF) interface, a scalable protocol for accessing and querying RDF data on the Web (Verborgh et al. 2016). The flexibility of Hydra makes it possible for servers to implement and/or extend that interface in many various ways, without breaking existing clients. What's more, Verborgh (2016) demonstrated that, to some extent, some server extensions can even benefit old clients that are not aware of these extensions.

Our MTMS kTBS is designed as a RESTful service using RDF as its internal data-model, so we had to deal with problems very similar to those that would later be addressed by LDP and Hydra (which then encouraged me to join both groups). To address those problems, we proposed the RDF-REST framework (Champin 2013; Médini et al. 2014). In RDF-REST, all resources (local or remote) are handled the same way, through a uniform interface (which translates seamlessly to HTTP verbs in the case of remote resources). Resource representations are mapped to RDF graphs, regardless of their original format, thanks to an extensible library of parsers and serializers; this feature allows RDF-REST applications to inter-operate with third-party services. Note that RDF-REST takes into account ambivalence, as different RDF-REST applications could have different interpretations of the data published by third-party services. This would only require them use a different set of parsers/serializers.

This framework could easily be used to implement LDP servers, even if it does not enforce LDP compliance (it allows to implement different interaction models). It would even more easily inter-operate with other LDP servers, as they already provide RDF representations. It is equally easy to publish a Hydra API documentation for an RDF-REST service, although currently this documentation has to be manually edited, independently from the code. In a future version of RDF-REST, we plan to reverse this workflow, by requiring service developers to write an API documentation, and using it to drive the behavior the service (at least the part captured by the Hydra vocabulary – for more specific behaviors, the developer will still have to code). This will prevent duplication of code and guarantee the compliance of the API documentation with the actual implementation of the service.

Throughout the works (ours and others) presented in this chapter, it can be seen how support for ambivalence and multiple interpretations is a recurring concern in Web based applications. It can not be otherwise given the large scale and the inherent openness of the Web. Moreover, this challenge is constantly renewed by the fast evolution of Web technologies, as much as that of the uses of the Web.

Notes

[1] http://www.netflixprize.com/
[2] http://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=75
[3] https://duckduckgo.com/privacy
[4] https://www.qwant.com/privacy
[5] To choose, quoting Pariser's metaphor, between "information vegetable" and "information dessert".
[6] http://liris.cnrs.fr/coatcnrs/wiki/doku.php
[7] http://tbs-platform.org/tbs/doku.php/tools:traceme
[8] *(1, 2)* https://github.com/adlnet/xAPI-Spec/
[9] In fact, the RESTful protocol defined by the Tin Can API is very similar to the one used by kTBS, so we are confident that this integration should not be too hard to achieve.
[10] http://tbs-platform.org/tbs/doku.php/tools:samotraceme
[11] http://tbs-platform.org/tbs/doku.php/tools:taaabs
[12] http://hubblelearn.imag.fr/
[13] An acrostic is a poem (or other form of writing) in which the first letter (or syllabe or word) of each line (or paragraph, or other recurring feature in the text) spells out a word or a message. (from Wikipedia)
[14] This kinship between presentation and interpretation will be further discusses in Chapter 6.
[15] http://microformats.org/
[16] http://schema.org/
[17] http://programmableweb.com/
[18] It is true that, in many cases, the user is asked for permission before an application shares their data with another one. However, users can only have this choice if both application providers have a prior agreement, and furthermore, nothing prevents them (technically, at least) to share this information without the user's consent.

Chapter bibliography

Abascal, Rocio, Michel Beigbeder, Aurélien Bénel, Sylvie Calabretto, Bertrand Chabbat, Pierre-Antoine Champin, Noureddine Chatti, et al. 2003. "Modéliser La Structuration Multiple Des Documents." In *Créer Du Sens à l'ère Numérique (H2PTM'03)*, 253–57. Hermes. http://liris.cnrs.fr/publis/?id=993.

Abascal, Rocio, Aurélien Bénel, Michel Beigbeder, Sylvie Calabretto, Bertrand Chabbat, Pierre-Antoine Champin, Noureddine Chatti, et al. 2004. "Un Modèle de Document à Structures Multiples." In *Sciences of Electronic, Technology of Information and Telecommunications (SETIT2004)*, 00. http://liris.cnrs.fr/publis/?id=1254.

Adida, Ben, and Mark Birbeck. 2008. "RDFa Primer." W3C Working Group Note. W3C. http://www.w3.org/TR/xhtml-rdfa-primer/.

Baker, Mark. 2008. "Hypermedia in RESTful Applications." *InfoQ* (blog). January 28, 2008. https://www.infoq.com/articles/mark-baker-hypermedia.

Barazzutti, Pierre-Loup, Amélie Cordier, and Béatrice Fuchs. 2015. "Transmute: An Interactive Tool for Assisting Knowledge Discovery in Interaction Traces." Research Report. Universite Claude Bernard Lyon 1 ; Universite Jean Moulin Lyon 3. https://hal.archives-ouvertes.fr/hal-01172013.

Barhamgi, Mahmoud, Pierre-Antoine Champin, Djamal Benslimane, and Aris M. Ouksel. 2007. "Composing Data-Providing Web Services in P2P-Based Collaboration Environments." In *Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings*, edited by John Krogstie, Andreas L. Opdahl, and Guttorm Sindre, 4495:531–545. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-540-72988-4_37.

Berners-Lee, Tim, Wendy Hall, James A. Hendler, Kieron O'Hara, Nigel Shadbolt, and Daniel J. Weitzner. 2006. "A Framework for Web Science." *Found. Trends Web Sci.* 1 (1): 1–130. https://doi.org/10.1561/1800000001.

Bonneau, Joseph, Elie Bursztein, Ilan Caron, Rob Jackson, and Mike Williamson. 2015. "Secrets, Lies, and Account Recovery: Lessons from the Use of Personal Knowledge Questions at Google." In *24th International Conference on World Wide Web*, 141–150. Florence, Italy: ACM. http://dl.acm.org/citation.cfm?id=2736277.2741691.

Buffat, Marc, Alain Mille, and Marco Picasso. 2015. "Feedbacks on MOOCS." In *CANUM 2014*, edited by Franck Boyer, Thierry Gallouet, Raphaèle Herbin, and Florence Hubert, 50:66–80. Carry-le-Rouet, FR: ESAIM: Proceedings and Surveys. https://doi.org/10.1051/proc/201550004.

Cazenave-Lévêque, Raphaël. 2016. “Interoperability among Trace Formalisms.” Master’s Thesis, Université Lyon 1.

Champin, Pierre-Antoine. 2013. “RDF-REST: A Unifying Framework for Web APIs and Linked Data.” In *Services and Applications over Linked APIs and Data (SALAD), Workshop at ESWC*, 1056:10–19. CEUR. Montpellier, France: CEUR Workshop Proceedings. https://hal.archives-ouvertes.fr/hal-00921662.

De Nies, Tom, Frank Salliau, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. 2015. “TinCan2PROV: Exposing Interoperable Provenance of Learning Processes through Experience API Logs.” In *Proceedings of the 24th International Conference on World Wide Web Companion*, 689–94. ACM Press. https://doi.org/10.1145/2740908.2741744.

Estrin, Deborah. 2014. “Small Data, Where n = Me.” *Commun. ACM* 57 (4): 32–34. https://doi.org/10.1145/2580944.

Fielding, Roy Thomas. 2000. “Architectural Styles and the Design of Network-Based Software Architectures.” Doctoral dissertation, University of California, Irvine. http://www.ics.uci.edu/%7Efielding/pubs/dissertation/top.htm.

Glazkov, Dimitri. 2016. “Custom Elements.” W3C Working Draft. W3C. https://www.w3.org/TR/custom-elements/.

Goel, Kavi, and Pravir Gupta. 2011. “Introducing Schema.Org: Search Engines Come Together for a Richer Web.” *Official Google Webmaster Central Blog* (blog). June 2, 2011. https://webmasters.googleblog.com/2011/06/introducing-schemaorg-search-engines.html.

Hasan, Rakebul. 2014a. “Generating and Summarizing Explanations for Linked Data.” In *The Semantic Web: Trends and Challenges (Proceedings of ESWC 2014)*, 473–487. Springer. https://doi.org/10.1007/978-3-319-07443-6_32.

———. 2014b. “Predicting Query Performance and Explaining Results to Assist Linked Data Consumption.” Phd Thesis, Université de Nice-Sophia Antipolis.

Heflin, Jeff, James Hendler, and Sean Luke. 1999. “SHOE: A Knowledge Representation Language for Internet Applications.” http://drum.lib.umd.edu/handle/1903/1044.

Kong Win Chang, Bryan, Marie Lefevre, Nathalie Guin, and Pierre-Antoine Champin. 2015. “SPARE-LNC : Un Langage Naturel Contrôlé Pour l’interrogation de Traces d’interactions Stockées Dans Une Base RDF.” In *IC2015*. Rennes, France: AFIA. https://hal.archives-ouvertes.fr/hal-01164383.

Kramer, Adam D.I. 2012. “The Spread of Emotion via Facebook.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 767–770. CHI ’12. New York, NY, USA: ACM. https://doi.org/10.1145/2207676.2207787.

Lanthaler, Markus, and Christian Gütl. 2013. “Model Your Application Domain, Not Your JSON Structures.” In *Proceedings of the 22Nd International Conference on World Wide Web*, 1415–1420. WWW ’13 Companion. New York, NY, USA: ACM. https://doi.org/10.1145/2487788.2488184.

Lebo, Timothy, Satya Sahoo, and Deborah McGuinness. 2013. “PROV-O: The PROV Ontology.” W3C Recommendation. W3C. https://www.w3.org/TR/prov-o/.

Lee, WonSuk, Werner Bailer, Tobias Bürger, Pierre-Antoine Champin, Jean-Pierre Evain, Véronique Malaisé, Thierry Michel, et al. 2012. “Ontology for Media Resources 1.0.” W3C Recommendation. W3C. http://www.w3.org/TR/mediaont-10/.

Mansour, Essam, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. “A Demonstration of the Solid Platform for Social Web Applications.” In *Proceedings of the 25th International Conference Companion on World Wide Web*, 223–226. International World Wide Web Conferences Steering Committee. http://dl.acm.org/citation.cfm?id=2890529.

Marcotte, Ethan. 2010. “Responsive Web Design.” *A List Apart* (blog). May 25, 2010. http://alistapart.com/article/responsive-web-design.

Médini, Lionel, Pierre-Antoine Champin, Michaël Mrissa, and Amélie Cordier. 2014. “Towards Semantic Resource Mashups.” In *Services and Applications over Linked APIs and Data (SALAD), Workshop at ESWC*, 6–9. CEUR Workshop Proceedings. http://liris.cnrs.fr/publis/?id=6685.

Moreau, Luc, and Paolo Missier. 2013. “PROV-DM: The PROV Data Model.” W3C Recommendation. W3C. http://www.w3.org/TR/prov-dm/.

Pariser, Eli. 2011. *Beware Online “Filter Bubbles.”* TED Talk. https://www.ted.com/talks/eli_pariser_beware_online_filter_bubbles.

Pieters, Simon. 2016. “WebVTT: The Web Video Text Tracks Format.” Draft Community Group Report. W3C. https://w3c.github.io/webvtt/.

Sanderson, Robert, Paolo Ciccarese, and Benjamin Young. 2016. “Web Annotation Data Model.” W3C Candidate Recommendation. W3C. https://www.w3.org/TR/annotation-model/.

Snell, J. M., M. Atkins, W. Norris, C. Messina, M. Wilkinson, and R. Dolin. 2011. “Activity Streams 1.0.” Activity Streams working group. http://activitystrea.ms/specs/json/1.0/.

Snell, James, and Evan Prodromou. 2016. “Activity Streams 2.0.” W3C Candidate Recommendation. W3C. http://www.w3.org/TR/activitystreams-core/.

Speicher, Steve, John Arwe, and Ashok Malhotra. 2015. “Linked Data Platform 1.0.” W3C Recommendation. W3C. http://www.w3.org/TR/ldp/.

Sporny, Manu, Gregg Kellogg, and Markus Lanthaler. 2014. “JSON-LD 1.0 – A JSON-Based Serialization for Linked Data.” W3C Recommendation. W3C. http://www.w3.org/TR/json-ld-syntax/.

Stegmaier, Florian, Werner Bailer, Tobias Bürger, Mari Carmen Suárez-Figueroa, Erik Mannens, Martin Höffernig, Pierre-Antoine Champin, Jean-Pierre Evain, Mario Döller, and Harald Kosch. 2013. “Unified Access to Media Metadata on the Web: TowardsInteroperability Using a Core Vocabulary.” *IEEE MultiMedia* 20 (2): 22–29. https://doi.org/10.1109/MMUL.2012.55.

Steiner, Thomas, Hannes Mühleisen, Ruben Verborgh, Pierre-Antoine Champin, Benoît Encelle, and Yannick Prié. 2014. “Weaving the Web(VTT) of Data.” In *Linked Data on the Web (LDOW2014)*, edited by Tim Berners-Lee Christian Bizer Tom Heath, Sören Auer, 1–10. http://liris.cnrs.fr/publis/?id=6501.

Styles, Rob, Nadeem Shabir, and Jeni Tennison. 2009. “A Pattern for Domain Specific Editing Interfaces Using Embedded RDFa and HTML Manipulation Tools.” In . Heraklion, Crete, Greece. https://confluence.ontotext.com/download/attachments/9504111/Talis+Aspire-+Pattern+for+Domain+Specific+Editing+Interfaces+Using+Embedded+RDFa+and+HTML+Manipulation+Tools+(SFSW+2009).pdf.

Troncy, Raphaël, Erik Mannens, Silvia Pfeiffer, and Davy Van Deursen. 2012. “Media Fragments URI 1.0 (Basic).” W3C Recommendation. W3C. http://www.w3.org/TR/media-frags/.

Verborgh, Ruben. 2016. “Querying History with Linked Data.” *Ruben Verborgh’s Blog* (blog). June 22, 2016. http://ruben.verborgh.org/blog/2016/06/22/querying-history-with-linked-data/.

Verborgh, Ruben, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. 2016. “Triple Pattern Fragments: A Low-Cost Knowledge Graph Interface for the Web.” *Journal of Web Semantics* 37–38 (March): 184–206. https://doi.org/doi:10.1016/j.websem.2016.03.003.

5. Multimedia annotation and hypervideo

Consider a book critic; while reading a book, she can annotate it by writing comments in the margins, dog-earing pages and/or using colored post-it notes as bookmarks. If the book is a digital file (*e.g.* a PDF document), her reading software provides similar functionalities, as well as additional ones, such as a dynamic table of contents reminding her at every moment of the overall structure of the book, showing which part she is currently reading, and allowing her to quickly navigate to another part. She can search the whole text of the book as well as her own annotations, jump to the corresponding location immediately, and just as easily jump back to the previous visited locations. Finally, she can easily copy-paste parts of the text in order to include them in her review.

Now consider a film critic; not so long ago, she still had to rely on a tape or DVD to watch the movie, with little possible interaction besides pause, rewind and fast-forward (and jumping to a specific chapter, in the case of DVDs). Nowadays, files and streaming have largely replaced tapes and DVDs, but software video players hardly provide more functionalities than their mechanical counterparts. Why has digitization not allowed audiovisual documents to evolve the way text has?

The main reason is probably that practices around audiovisual are much less mature than practices around text. Indeed, the technical means to capture and render videos are very recent (when compared to written text or still images), and even more so their availability to a large public. Furthermore, video is inherently more complex, as it has its own temporality, to which the reader must yield in order to access the audiovisual material. While it is possible to skim a text or glance at a photo, such things are not immediately possible with a video. Interestingly, when Nelson (1965) coined the term “hypertext”, he also proposed the notion of “hyperfilm”, described as “a browsable or vari-sequenced movie”, noticing that video and sound documents were currently restricted to linear strings mostly for mechanical reasons. Obviously, although hypertext (and more generally text-centered hypermedia) has become common place, Nelson’s vision of hyperfilms is not so widespread yet.

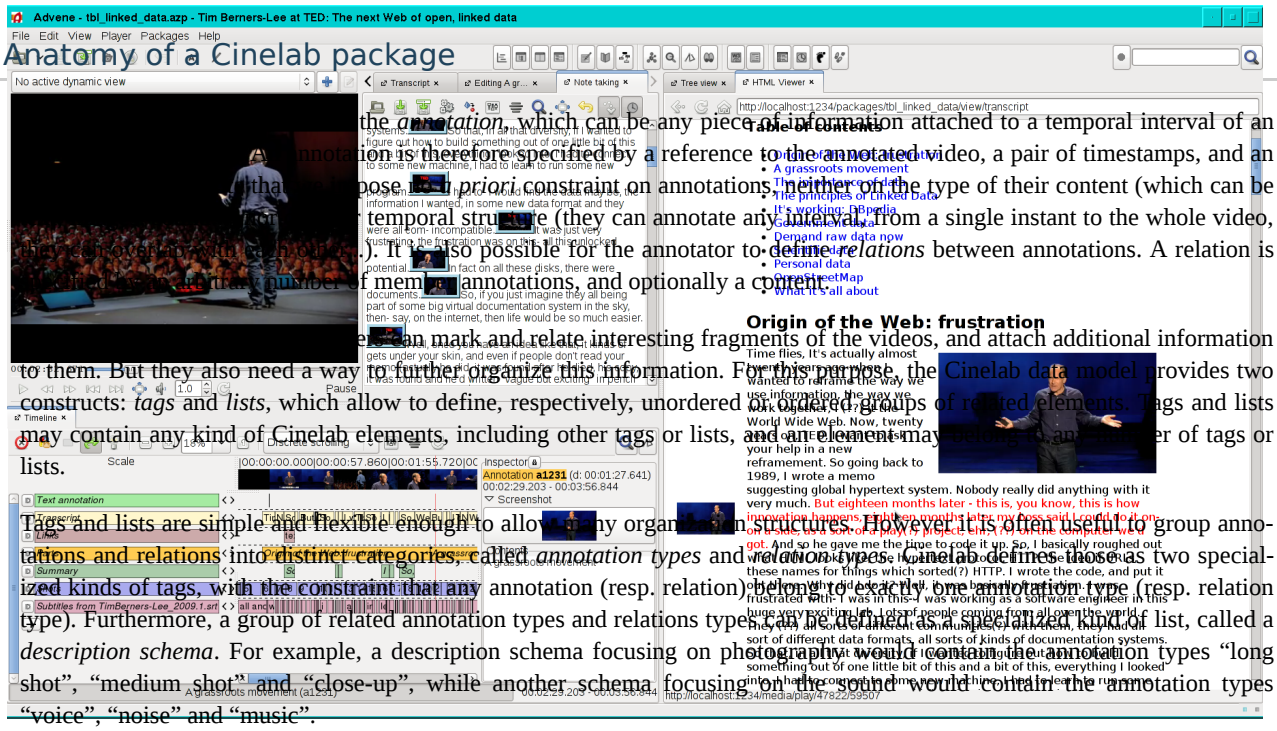
In this chapter, I will present our contributions in the topic of hypervideo based on video annotations. The first part will focus on our seminal work on Advene and the Cinelab data model. Then I will describe how video annotations often relate to traces, and how hypervideos can be used as a modality of MTBS. Finally, in the last section, I will show how various standards are converging to bring hypervideos to the Web.

5.1. Advene and the Cinelab Data Model

The Advene project[1] was born in 2002 out of this assessment that, although it was technically possible to improve the way an active reader may interact with videos, only basic tools were actually available, mostly because no well-established practice (such as bookmarking, annotation, *etc.*) existed yet for audiovisual documents, that could have set off the creation of better tools. To be fair, a few such tools did exist at the time (for example Anvil[2]), but those were generally very focused on a particular field and a particular task (for example behavior analysis in human sciences).

Our goal was therefore to build a generic and extensible platform, that would allow the exploration and stabilization of new practices for video active reading. Since such a platform would allow new forms of interactions with audiovisual material, it would also inevitably foster new documentary forms, so Advene is both a tool for active reading, and a hypermedia authoring platform.

In order to support the emergence of new active reading practices, Advene had to provide a versatile data model. This data model was refined over time, resulting to the Cinelab data model (Aubert, Champin, and Prié 2012).



Anatomy of a Cinelab package

the *annotation*, which can be any piece of information attached to a temporal interval of an annotation is therefore specified by a reference to the annotated video, a pair of timestamps, and an pose [illegible] *priori* constraint on annotations, neither on the type of their content (which can be r temporal structure (they can annotate anything from a single instant to the whole video,). It is also possible for the annotator to define *relations* between annotations. A relation is [illegible] number of members annotations, and optionally a content.

e can mark and relate interesting fragments of the videos, and attach additional information to them. But they also need a way to further organize this information. For this purpose, the Cinelab data model provides two constructs: *tags* and *lists*, which allow to define, respectively, unordered or ordered groups of related elements. Tags and lists may contain any kind of Cinelab elements, including other tags or lists, and an element may belong to any number of tags or lists.

Tags and lists are simple and flexible enough to allow many organization structures. However it is often useful to group annotations and relations into distinct categories, called *annotation types* and *relation types*. Cinelab defines those as two specialized kinds of tags, with the constraint that any annotation (resp. relation) belongs to exactly one annotation type (resp. relation type). Furthermore, a group of related annotation types and relation types can be defined as a specialized kind of list, called a *description schema*. For example, a description schema focusing on photography would contain the annotation types “long shot”, “medium shot” and “close-up”, while another schema focusing on [illegible] would contain the annotation types “voice”, “noise” and “music”.

It is important to understand that the information carried by annotations (and relations) is not bound *a priori* to any particular rendering. For example, consider annotations of type “voice” that contain the transcription of what is said in the annotated fragment of the video. They could obviously be displayed as subtitles (for the hearing impaired) but could also be displayed beside the video as an interactive and searchable transcript (as in the right side of Fig. 5.1); or they could be used indirectly by a hypervideo playing additional sounds (*e.g.* director’s comments or audio-descriptions for the visually impaired) to avoid overlapping with the characters speaking in the video. In this respect, annotations are analogous to HTML tags: they convey structure and semantics, and are orthogonal to presentation concerns.

Therefore, users also need to be able to specify and customize how they want their annotations to be presented, both during their activity, and afterwards to present the result of their work. In the Cinelab data model, a *view* is the specification of how to render a subset of the annotations. An application may support different kinds of views; in Advene, we distinguish

- *ad-hoc* views, which are specific configurations of the GUI components available in the application (for example, the graphical time-line at the left-bottom of Fig. 5.1),
- static views, which are XML or HTML documents generated through a dedicated template language (one of them illustrated on the right side of Fig. 5.1), and
- dynamic views, which are described by a list of Event-Condition-Action (ECA) rules, triggered while the video is playing (in order, for example, to overlay annotation content on the video, or automatically pause at the end of a given annotation).

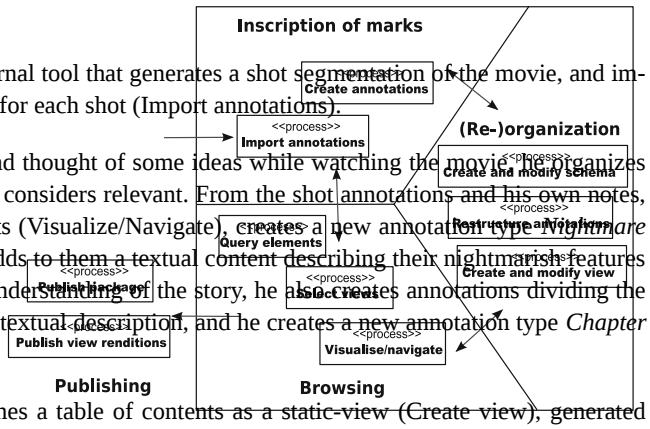
All the elements described above[3] are grouped in a file called a *package*, which can be serialized in different formats (we have defined an XML-based and a JSON-based format). As packages do not contain the annotated audiovisual material, but only references to it, they are generally small enough to be easily shared (on the Web, *via* e-mail or USB sticks). This was also meant to avoid issues related to copyrighted videos; it is the responsibility of each user of a package to gain legitimate access to the videos referenced in it. Note also that any package can *import* elements from other packages, in order to build upon somebody else’s work.

Active reading with Advene

Our experience with Advene suggests that video active reading can be decomposed into a number of processes, enumerated in Fig. 5.2. We also propose to group those processes in four intertwined phases: inscription of marks, (re-)organization, browsing and publishing. We have shown (Aubert et al. 2008) how those processes are supported by Advene and the Cinelab model, and how they can be mapped to the canonical processes identified by Hardman et al. (2008).

As an example, let us consider Mr Jones, teacher in humanities, who wants to give a course about the expression of mood in movies. He bases his work on the movie Nosferatu (Murnau 1929), and more specifically on how the movie’s nightmarish mood is built[4]. He starts from scratch, having seen the movie only once and read some articles about it.

Using the note-taking editor of Advene, Mr Jones types timestamped notes in a textual form, that he will later convert to annotations (Create annotations). He also uses an external tool that generates a shot segmentation of the movie, and imports the resulting data in Advene, generating one annotation for each shot (Import annotations).



Now that the teacher has created a first set of annotations, and thought of some ideas while watching the movie, he organizes the annotations in order to emphasize the information that he considers relevant. From the shot annotations and his own notes, Mr Jones identifies the shots containing nightmarish elements (Visualize/Navigate), creates a new annotation type *Nightmare* (Create schema), copies the relevant annotations into it and adds to them a textual content describing their nightmarish features (Create/Restructure annotations). As he has gained a better understanding of the story, he also creates annotations dividing the movie in chapters, each of them containing a title and a short textual description, and he creates a new annotation type *Chapter* for those annotations.

In order to ease the navigation in the movie, Mr Jones defines a table of contents as a static-view (Create view), generated from the annotations of type Chapter, illustrated by screenshots extracted from the movie, with hyperlinks allowing to play the corresponding part of the movie. He also creates a dynamic view that displays the title of the current chapter as a caption over the video, in order to always know which part of the movie is playing when he navigates through the annotations.

Taking advantage of all those annotations and the newly created views, the teacher wants to dig into some ideas about the occurrence of specific characters or animals in the movie. He can select the corresponding annotations manually by identifying them in a view (Visualize/Navigate), or use Advene's search functionalities to automatically retrieve a set of annotations meeting a given criterion (Query). Doing so, he identifies a number of shots featuring animals (spiders, hyenas...) that contribute to the dark mood of the movie.

While browsing the movie, he also creates new annotations (Create annotations) and new types (Create and modify schemas) as he notices other relevant recurring features. In the active-reading analysis, we find here a quick succession of browsing-inscription-organisation activities, when users decide to enrich the annotations while watching the movie. Inscription occurrences may last a couple of seconds, and should not be obtrusive to the browsing process.

This continuous cycle also brings Mr Jones to create more specific views, dedicated to the message/analysis that he wishes to carry: in order to have an overview of the nightmarish elements, he decides to create a view that generates a dynamic montage (Create view), chaining all the fragments annotated by the Nightmare type. This allows him to more precisely feel and analyze their relevance and relationships: watching his new montage (Select view/Visualize/Navigate) corroborates the ideas that he wishes to convey, and allows him to have a clearer view of how he will present them to his students.

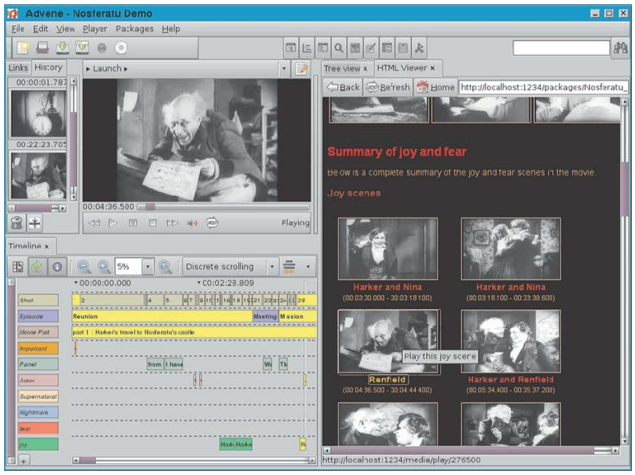


Fig. 5.3 An actual analysis of Nosferatu in Advene (Aubert et al. 2008)

Now that Mr Jones has identified the relevant items and refined his analysis, he can write it down as a critique, in a static view illustrated by screenshots of the movie linked to the corresponding video fragments (Create view). He prints the rendition of this static view from a standard web browser, in order to distribute it to his students in class (Publish view renditions). He also cleans up the package containing his annotations and views, removing intermediate remarks and notes, in order to keep only the final set of annotations, description schemas and views. He uploads that package to his homepage on a web-server (Publish package), so that his students can download it and use it to navigate in the movie. This second option is more constraining for the students, requiring them to use Advene or a compatible tool[5]. But on the other hand it allows them to pursue the analysis through the same active reading cycle as described above, without having to start from scratch.

From this use-case, it appears that Cinelab packages are very similar to the multi-structured documents presented in Section 4.2: the complex relations between the linear temporality of the video and the elements of a package allow multiple readings and interpretations. A subset of those interpretations is "materialized" by the views provided in the package, but it is not closed as long as the package is distributed as is, reusable and augmentable by others.

Finally, Mr Jones publishes a copy of his package after removing all annotations, leaving only the description schema he has defined and the associated generic views. As an assignment, he asks his student to analyze with Advene another horror movie, reusing the annotation structure provided in this "template" package (that they can import in their own package). This empha-

sizes another important feature of the Cinelab data model: not only does it allow the exploration of innovative annotation practices, it also encourages their *stabilization* and their sharing as explicit and reusable organization structures.

5.2. Videos and annotations as traces

There is an obvious similarity between the Cinelab model presented above and the Trace meta-model presented in Chapter 2. Indeed, videos are often used to record a situation, hence as a kind of raw traces. Video annotations, on the other hand, are a mean to describe the content of a video in a structured way, easier to handle and process that the audiovisual signal itself. As they provide a machine-readable description of the filmed situation, such annotations can therefore be considered as obsels, all the more that they have the same basic structural features: they are anchored to a time interval, typed, and potentially carrying additional information.

Conversely, any obsel about an activity that has been filmed can be synchronized with the corresponding video, hence considered as annotating that video. Following the principles explored with Advene and Cinelab, those obsels and the video can be used together to generate a hypervideo, which can in turn be used as a mean to visualize and interact with the trace. The trace model, defining the different types of obels that he trace may contain, plays a role very similar to the description schemas in Cinelab.

Video-assisted retrospection

In the *Ithaca* project[6] (2008-2011), we have studied the use of traces to support synchronous collaborative activities, as well as the duality of traces and video annotations. For this, we have developed a proptotype called Visu.

Visu is an online application with two parts. The first part is a virtual classroom (see Fig. 5.4) for a teacher and a group of students, offering a video-conferencing and chat platform as well as more specific functionalities with educational purposes: the teacher can define in advance a course outline and a set of documents (left side of the figure), that he/she can push in the chat during the session. The second part of Visu is called the retrospection room; it allows the teacher to play back the video of a past session. In *Ithaca*, the teachers using Visu were actually in training, and the retrospection room was used to help them understand and overcome the difficulties they may have encountered during the class.

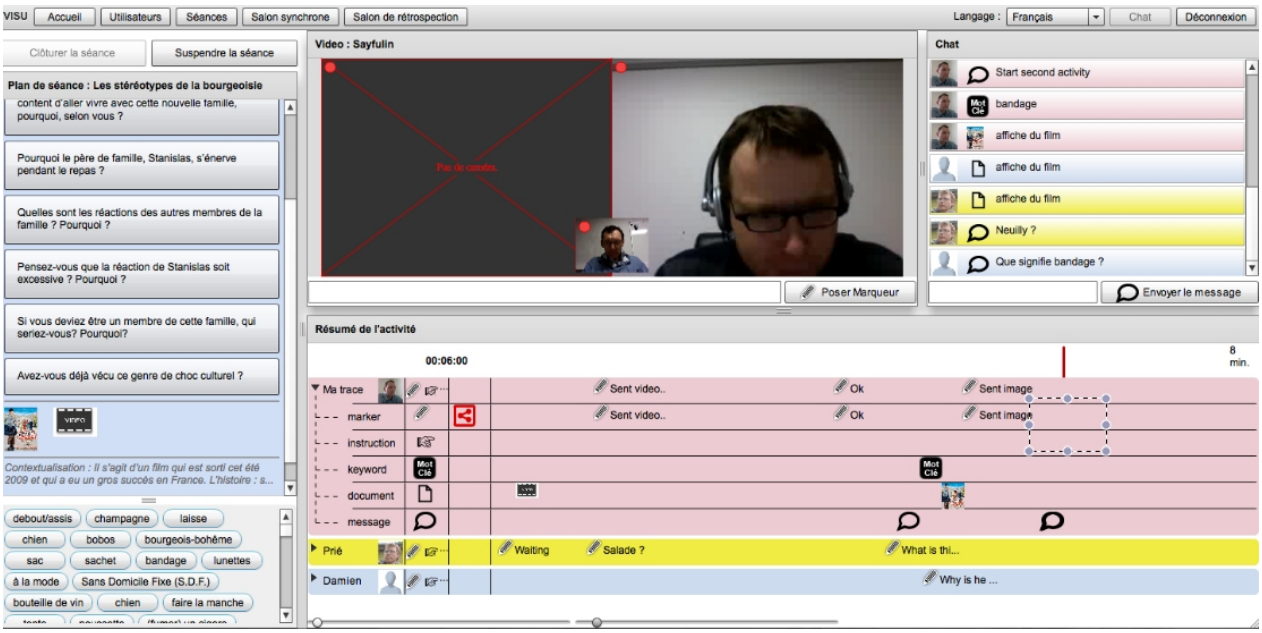


Fig. 5.4 The virtual classroom of Visu

During the session, every interaction of the users with the application (typing in the chat, pushing a document, opening a document, *etc.*) is traced and displayed on a graphical time-line (lower-part of Fig. 5.4). It provides the teacher and the students with a sense of reflexivity on the group's activity. Information in the time-line can also be used by the teacher at the end of the session to do a short debriefing with the students. In the retrospection room, the time-line is synchronized with the video playback, and gives the teacher a more complete view of what happened during the session.

In addition to those automatically collected annotations, Visu allows the teacher and the students to manually add markers in the time-line, containing short messages. Unlike chat messages, which are public and used to communicate synchronously

with the group, markers are generally only visible by the teacher, and used to come back later (*i.e.* during the debriefing or the retrospection) to the moments of the session when they were created.

In the retrospection room, the teacher can further annotate the video with a third kind of annotation called “comments”. Contrarily to the annotations created during the session, comments are not restricted to annotate a single instant of the video, but may span a time interval. They are used to synthesize the hindsight gained by the training teacher in the retrospection room. It is also possible to produce a report by reorganizing the comments, adding more text and screenshots from the video. This functionality can be compared to the trace-based redocumentation process (Yahiaoui et al. 2011) presented in the third chapter.

Visu has been used in different settings, and an analysis of our first experiments was presented by Betrancourt, Guichon, and Prié (2011). Although the application may first be cognitively challenging, the teachers got used to it after the first session, and did use its specific functionalities (especially markers). Moreover, it has been observed that different teachers had developed different practices with markers, which confirms the flexibility of that functionality. Finally, the teachers used less markers during the last session, which tends to indicate that one motivation for using markers was to prepare for future sessions, using the retrospection room.

Archiving and heritage

Video traces are not limited to short-term usage, they can be archived to serve as cultural heritage, in which case it is critical to index them effectively, to ensure their usability in the long run. One of the goals of *Spectacle en ligne(s)* project[7], presented in our paper by Ronfard et al. (2015), was to propose such effective indexing structures, based on the Cinelab model.

More precisely, this project aimed at creating a richly indexed corpus of filmed performing arts, and exploring innovative uses for the performers themselves, other professionals, and the larger public. Two shows were covered by that project: *Cat on a hot tin roof* by Tenessee Williams, directed by Claudia Stavisky at the Théatre des Célestins (Lyon), and the baroque opera *Elena* by Francesco Cavalli, directed by Jean-Yves Ruf and conducted by Leonardo García Alarcón at the Aix-en-Provence Festival. The originality of the created corpus was that we didn’t capture the public performances, instead we recorded all the rehearsals.

Fig. 5.5 Three steps in the theater rehearsal process for *Cat on a hot tin roof*: table readings, early rehearsals in a studio, late rehearsals on stage.

As rehearsals are typically private moments, sometimes even qualified as “sacred”, the setting for capturing them was designed to be as unintrusive as possible. A dedicated operator had the responsibility of recording each rehearsal, using a fixed full-HD camera controlled by a PC laptop, that also allowed them to annotate the video while it was being captured. More precisely, the embedded application was designed around a predefined description schema that had been created specifically for that project. The role of those annotations was to provide a first level of indexation: a rehearsal is segmented with annotations of two different types, *Performance* and *Discussion*. Each chapter is then described with a number of properties: which part of the play/opera was being rehearsed, which actors where presents, was it with or without costumes, with or without sets, *etc.* A third annotation type, *Moment of interest*, allowed to further describe specific instants of the rehearsals with a free-text comment and some categories, defined on the fly by the operator. In the end, 419 hours of video were captured, and 10,498 annotations were created.

But the annotation process didn’t stop at the end of the rehearsal. First, the annotations created during the session required some off-line manual cleansing (correcting misspellings, harmonizing categories...). Then, some partners in the project have proposed automatic annotation processes, based on the audiovisual signal and the cues provided by the manual annotations. For example, using machine learning techniques to recognize the voice and appearance of each actor, it was possible to create fine-grained annotations aligning the video with individual lines of the script, and annotations spatially locating each actor in each frame of the video. This is computationally very expensive, so we could only process a subset of the corpus in the timeframe of the project, but the results were very encouraging (Gandhi and Ronfard 2013).

To demonstrate the benefit of this annotated corpus, we have developed a number of prototypes. The whole corpus can be searched online[7], with a faceted browser (based on the features describing each chapter, and the categories of the moments of interest). Each video can be watched, augmented with a time-line displaying the annotations (similar to the one of Advene), and a synchronized script, automatically scrolling to the part being rehearsed (Fig. 5.6). This allows critics, teachers and other professionals to study the creative process in an unprecedented way. On the scenes where we have computed line-level annotations, each line is highlighted in the script when it is delivered, and it is possible to navigate directly to the same line in any other rehearsal. Using the spatial location of the actors in the video we have simulated multiple cameras, each of them following one character (by simply zooming on the corresponding area of the original video). This was made possible by the high resolution of the original video. Then, using the line annotations, we have proposed a virtual montage by automatically switching to the character currently speaking, making the video less monotonous to watch. We have also considered alternative ways to generate such a virtual montage, like framing two characters instead of one during fast paced dialogues.

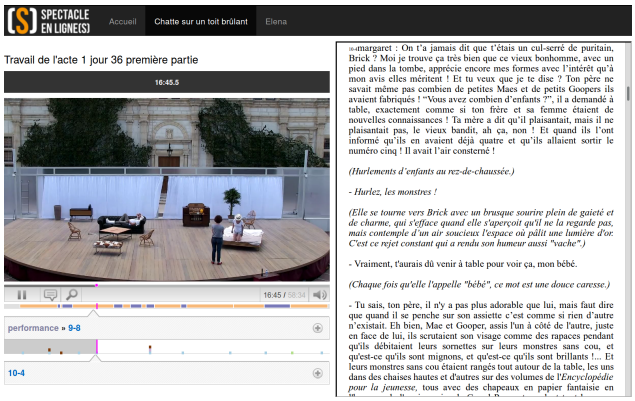


Fig. 5.6 The Spectacle en ligne(s) platform

Interestingly, during the production of the archive, the creative crew reacted quite positively to the experiment and expressed interest in getting immediate feedback. While this had not been planned, we started designing mobile applications that they could tentatively use for (i) viewing on their smartphones the live feed being recorded and (ii) adding their own (signed) annotations and comments collaboratively. While not fully implemented, this feature was presented to the directors and their collaborators as mock-ups. These mock-ups were generally well received and are likely candidates as an addition to the existing system for future experiments.

Beyond *Spectacle en ligne(s)*, we are involved in another project concerned with video archives and cultural heritage. The former prison of Montluc, in Lyon, has been turned into a memorial in 2010. This memorial focuses on the period when this prison was used by the Nazis during World War II. A research group in sociology has conducted an inquiry to analyze and document this heritage process, shedding light on other memorable periods where that prison was used. From this inquiry, they produced a corpus of video interviews and additional materials (photos, documents). Their goal was to publish it in order to sustain the continuous emergence of multiple memories related to Montluc, beyond the one highlighted by the memorial itself. Therefore, their challenge was to make this multiplicity of histories and memories legible, to allow multiple interpretations of the place by people with different experiences and pasts, and to encourage novel uses of the venue. In collaboration with them, we have designed a Web application[8] providing access to this corpus (Michel et al. 2016), but also allowing users to add their own annotations, keeping the heritage process in constant momentum. In the future, we plan to study the interaction traces of the users, as well as the annotations they contributed, to evaluate and improve the design of the Web application with respect to those goals.

5.3. Hypervideos on the Web

When we started working on Advene (in 2002), hypervideo in general, and video integration with the Web in particular were still in their infancy. At that time, the main way to integrate a video player in a HTML page was to use a proprietary browser plugin (very much frowned upon), and none of the popular video sharing websites, that are now an integral part of the Web ecosystem, existed yet.

Still, from the very beginning, we aimed to integrate Advene with Web technologies as much as possible. As explained in Section 5.1, static views in Advene produce XML or HTML documents, which can be exported and published on any Web server, but also delivered dynamically by a HTTP server embedded in the application. The benefit of that embedded server is that it has access to the annotated video. It can for example extract content on the fly (such as snapshots) that will be included in the static view, but most importantly, it can control the video player included in the Advene GUI. Advene thus exposes a number of URLs that can be used to drive its video player from any HTML page. For example, in Fig. 5.1, the HTML transcript on the right side is dynamic: every sentence is a link that will start the video player at that point of the talk. Although this is not a full-Web solution (it requires Advene to run on the client's machine) it allowed us to experiment very early-on with the interactions between video and HTML-based hypermedia.

In 2009, we started the ACAV project in partnership with Eurecom[9] and Dailymotion[10]. The goal was to improve the accessibility of videos for people with visual and hearing disabilities, using annotations to enrich videos in various ways, adapted to the viewers' impairment and to their preferences. For example, the descriptions for visually impaired people may be rendered on a braille display or as an audio-description (using a speech synthesizer), it may be more or less detailed, *etc.* The flexibility

offered by the Cinelab data model could be leveraged to achieve this goal. We have defined a description schema, specifying the different types of annotations required to enrich the video in the various ways required by impaired users, and we have prototyped a number of views using those annotations. The intended workflow is described in Fig. 5.7: signal-processing algorithms automatically produce a first set of annotations, which is then manually corrected and augmented. Two kinds of users were expected to contribute to those annotations: associations and enthusiasts concerned with disabilities and accessibility, and institutional video contributors, bound by legal obligations to make their videos accessible. Unfortunately, despite encouraging results with our prototypes (Champin et al. 2010; Villamizar et al. 2011), Dailymotion didn't go as far as put the system in production.

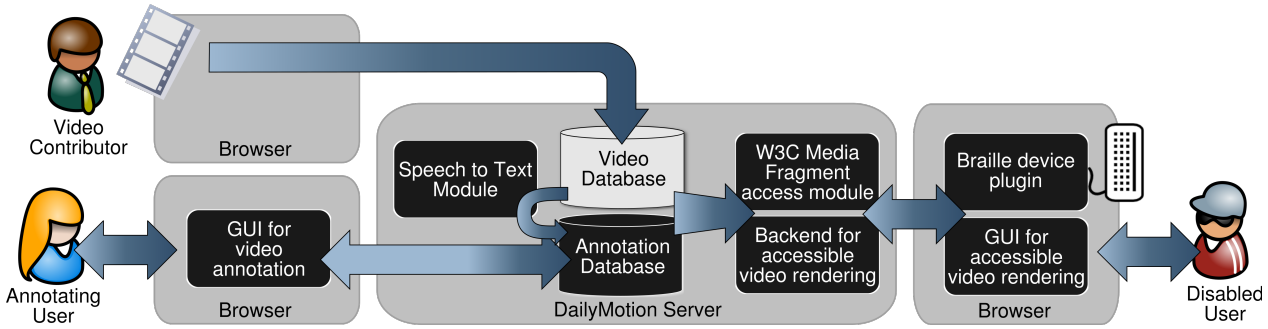


Fig. 5.7 The ACAV workflow for producing accessible hypervideos (Champin et al. 2010)

As video was increasingly becoming a first-class citizen of the Web, it also became possible to refine the notion of view in Cinelab, in order to align it with emerging technologies such as the HTML5 `video` tag. CHM (Sadallah, Aubert, and Prié 2011, 2014) is a generic component-based architecture for specifying Cinelab views, with an open-source implementation based on HTML5. Compared to Advene's templates and ECA-rules, CHM provides a more declarative way to describe hypervideos, and incorporate high-level constructs for the most common patterns, such as subtitles, interactive table of contents or interactive transcripts. Then, we have put those ideas one step further (Steiner et al. 2015)[11] by relying on the emerging *Web Components* standard (Glazkov 2016). With this new specification, it becomes possible to extend HTML with new tags, making hypervideo components even more integrated with Web standards, and easier to use by Web developers.

Finally, after video, annotations themselves are in the process of being standardized. With Media Fragment URIs (Troncy et al. 2012) we have a standard way to identify and address fragments of any video with its own URI, and a few proposals already exist to extend this recommendation[12][13] in a possible reactivation of the working group. Besides, the candidate recommendation by Sanderson, Ciccarese, and Young (2016) proposes a Web Annotation Data Model, which allows to annotate any Web resource or fragment thereof with arbitrary data (and possibly other resources), and to serialize those annotations as Linked Data (using JSON-LD). Our own work follows the same line as those standards. Our re-interpretation of WebVTT as Linked Data (Steiner et al. 2014) presented in Section 4.2 made use of Media Fragment URIs; although we defined a specific data-model (based on the structure of WebVTT), adapting it to Web Annotation should be relatively straightforward. Later, in the *Spectacle en ligne(s)* project presented above, we published the whole corpus of annotations as Linked Data (Steiner et al. 2015), after proposing an RDF vocabulary capturing the Cinelab data model. At the time, we aligned that vocabulary with some terms of the Web Annotation Data Model, but as the latter is about to become a recommendation, it would be interesting to update and refine this alignment.

In the longer term, we will probably redefine the Cinelab data model itself as an *extension* of the Web Annotation data model. Indeed, the overlapping concepts are close enough, so Cinelab annotations can ambivalently be re-interpreted as a special case of Web annotations. Both models would benefit from this unification, as Cinelab-aware tools (such as Advene) would become usable to publish standard-compliant annotations, and hence attractive to a larger audience.

Probably more than any other type of information, multimedia content lends itself to multiple interpretations. This is why the languages and tools used to handle this kind of content must be flexible enough. The works presented in this chapter describe our efforts to propose such languages and tools, not only by enabling subjective analyses to be expressed, but also by allowing to stabilize interpretative frameworks as sharable schemas.

Notes

[1] http://advene.org/
[2] http://www.anvil-software.org/
[3] Actually, the Cinelab model defines a few other categories of elements, which are not described here for the sake of conciseness and clarity. The interested reader can refer to the complete documentation (Aubert, Champin, and Prié 2012) for full details.
[4] Although this example is fictional, an actual Advene package corresponding to what Mr Jones would have produced can be downloaded at http://advene.org/examples.html, and is illustrated in Fig. 5.3.
[5] For example, the Institut de Recherche et d'Innovation (IRI) has adopted Cinelab for their own video annotation tools: http://www.iri.centrepompidou.fr/.
[6] https://liris.cnrs.fr/ithaca/
[7] *(1, 2)* http://spectacleenlignes.fr
[8] http://patrimonum.fr/montluc/
[9] http://eurecom.fr/
[10] http://dailymotion.com/
[11] https://github.com/tomayac/hyper-video
[12] http://olivieraubert.net/dynamic-media-fragments/
[13] http://tkurz.github.io/media-fragment-uris-ideas/

Chapter bibliography

Aubert, Olivier, Pierre-Antoine Champin, and Yannick Prié. 2012. "Cinelab Data Model." July 2012. http://advene.org/cinelab/.

Aubert, Olivier, Pierre-Antoine Champin, Yannick Prié, and Bertrand Richard. 2008. "Canonical Processes in Active Reading and Hypervideo Production." *Multimedia Systems Journal* 14 (6): 427–33. https://doi.org/10.1007/s00530-008-0132-2.

Betrancourt, Mireille, Nicolas Guichon, and Yannick Prié. 2011. "Assessing the Use of a Trace-Based Synchronous Tool for Distant Language Tutoring." In *Computer-Supported Collaborative Learning*, 1:478–85. Hong Kong, China. https://hal.archives-ouvertes.fr/hal-00806428.

Champin, Pierre-Antoine, Benoît Encelle, Nicholas W. D. Evans, Magali Ollagnier-Beldame, Yannick Prié, and Raphaël Troncy. 2010. "Towards Collaborative Annotation for Video Accessibility." In *7th International Cross-Disciplinary Conference on Web Accessibility (W4A 2010)*. https://doi.org/10.1145/1805986.1806010.

Gandhi, Vineet, and Rémi Ronfard. 2013. "Detecting and Naming Actors in Movies Using Generative Appearance Models." In *CVPR 2013 - International Conference on Computer Vision and Pattern Recognition*, 3706–13. Portland, Oregon, United States: IEEE. https://doi.org/10.1109/CVPR.2013.475.

Glazkov, Dimitri. 2016. "Custom Elements." W3C Working Draft. W3C. https://www.w3.org/TR/custom-elements/.

Hardman, Lynda, Željko Obrenović, Frank Nack, Brigitte Kerhervé, and Kurt Piersol. 2008. "Canonical Processes of Semantically Annotated Media Production." *Multimedia Systems* 14 (6): 327–40. https://doi.org/10.1007/s00530-008-0134-0.

Michel, Christine, Marie-Thérèse Têtu-Delage, Pierre-Antoine Champin, and Laetitia Pot. 2016. "Stimulating the Heritage Process with Web Platforms." *Les Cahiers Du Numérique*, Médiation des mémoires en ligne, 12 (3): 31–50. https://doi.org/10.3166/lcn.12.3.31-50.

Murnau, Friedrich Wilhelm. 1929. *Nosferatu*. Centraal Bureau voor Ligafilms.

Nelson, Theodor Holm. 1965. "Complex Information Processing: A File Structure for the Complex, the Changing and the Indeterminate." In *Proceedings of the 1965 20th National Conference*, 84–100. ACM '65. New York, NY, USA: ACM. https://doi.org/10.1145/800197.806036.

Ronfard, Rémi, Benoit Encelle, Nicolas Sauret, Pierre-Antoine Champin, Thomas Steiner, Vineet Gandhi, Cyrille Migniot, and Florent Thiery. 2015. "Capturing and Indexing Rehearsals: The Design and Usage of a Digital Archive of Performing Arts." In , 533–40. IEEE. https://doi.org/10.1109/DigitalHeritage.2015.7419570.

Sadallah, Madjid, Olivier Aubert, and Yannick Prié. 2011. "Component-Based Hypervideo Model: High-Level Operational Specification of Hypervideos." In *Document Engineering 2011 (DocEng 2011)*, edited by ACM, 53–56. http://liris.cnrs.fr/publis/?id=5290.

———. 2014. "CHM: An Annotation- and Component-Based Hypervideo Model for the Web." *Multimedia Tools and Applications* 70 (2): 869–903. https://doi.org/10.1007/s11042-012-1177-y.

Sanderson, Robert, Paolo Ciccarese, and Benjamin Young. 2016. "Web Annotation Data Model." W3C Candidate Recommendation. W3C. https://www.w3.org/TR/annotation-model/.

Steiner, Thomas, Hannes Mühleisen, Ruben Verborgh, Pierre-Antoine Champin, Benoît Encelle, and Yannick Prié. 2014. "Weaving the Web(VTT) of Data." In *Linked Data on the Web (LDOW2014)*, edited by Tim Berners-Lee Christian Bizer Tom Heath, Sören Auer, 1–10. http://liris.cnrs.fr/publis/?id=6501.

Steiner, Thomas, Rémi Ronfard, Pierre-Antoine Champin, Benoît Encelle, and Yannick Prié. 2015. "Curtains Up! Lights, Camera, Action! Documenting the Creation of Theater and Opera Productions with Linked Data and Web Technologies." In . https://hal.inria.fr/hal-01159826/document.

Troncy, Raphaël, Erik Mannens, Silvia Pfeiffer, and Davy Van Deursen. 2012. “Media Fragments URI 1.0 (Basic).” W3C Recommendation. W3C. http://www.w3.org/TR/media-frags/.

Villamizar, José Francisco Saray, Benoît Encelle, Yannick Prié, and Pierre-Antoine Champin. 2011. “An Adaptive Videos Enrichment System Based On Decision Trees For People With Sensory Disabilities.” In *8th International Cross-Disciplinary Conference on Web Accessibility (W4A 2011)*. https://doi.org/10.1145/1969289.1969299.

Yahiaoui, Leila, Yannick Prié, Zizette Boufaïda, and Pierre-Antoine Champin. 2011. “Redocumenting Computer Mediated Activity from Its Traces: A Model-Based Approach for Narrative Construction.” *Journal of Digital Information (JoDI)* 12 (3). https://journals.tdl.org/jodi/index.php/jodi/article/view/2088.

6. A proposal for ambivalent semantics

Every piece of information (document, knowledge base, *etc.*) can be interpreted in various ways. The works presented in this dissertation aim at taking into account, or even take advantage of this **ambivalence**. In contrast, many works in computer science, especially in the field of knowledge representation, consider that a formal description must have only one valid interpretation. This constraint is meant to guarantee the consistency of how that description is processed, and interoperability of the tools processing it. This view seems to conflate *ambiguity* with ambivalence, the former being obviously to avoid, but not at the cost of the latter.

In this chapter, I propose an alternative point of view on the notion of semantics, in an attempt to formalize ambivalence rather than exclude it.

6.1. Terminology

Language and sentences

We set this proposal in the context of language theory. We define a **language** as a set of **sentence**s. The sentences of a language are not atomic elements, but can be described as a combination of **term**s, taken from a set called the **vocabulary** of that language. Unlike classical language theory, we are not limiting the structure of sentence to sequences of terms: we include for example in our proposal tree grammars (Nivat and Podelski 1992) and graph grammars (Rozenberg 1997). This inclusive definition of languages allows us to capture data actually processed by machines (sequences of discrete symbols) as well as more abstract structures represented by these data.

Here are a few examples of languages:

- Every set of terms can be considered as a trivial language (where each sentence is made of a single term); for example, the language of boolean values or the language of all integers.
- The language of all unicode strings: its vocabulary is the set of all unicode characters (The Unicode Consortium 2016), its sentences are finite sequences of those characters.
- The language of XML trees: its vocabulary is the set of unicode strings, its sentences are partially ordered trees, whose nodes are typed (element, attribute, text or comment) and labelled with terms of the vocabulary – with constraints on the strings labeling certain types of nodes (Cowan and Tobin 2004).
- The language of RDF graphs: its vocabulary is the set of all IRIs, literals a blank node identifiers; its sentences are graphs whose nodes are labelled by terms, and whose arcs are labelled with IRIs (Schreiber and Raimond 2014).

Note also that a language can be defined as a subset of another language. For example:

- the Python programming language is a subset of the language of unicode strings;
- the language XHTML (Pemberton 2000) is a subset of the language of XML trees.

Meaning and interpretation

To talk about the semantics of languages, we first define the notions of meaning and interpretation.

The **meaning** of a sentence is a non-formal property that is ascribed to that sentence by an external (human) observer. Such an extrinsic meaning can therefore not be unique for a given sentence: it depends on the observer, on their situation, *etc.* We no-

tice incidentally that the term “meaning” itself carries a notion of *intention* (as in “I didn’t mean to do that”), and that this proximity can also be found it its french translation *vouloir dire* (literally “to want to say”).

We call **interpretation** of a language L a partial function from L to another language L'. The inverse relation of an interpretation is sometimes called a **representation**: if $I : L \to L'$ is an interpretation function, and x is a sentence of L, then $y = I(x)$ is the interpretation of x under I, and x is a representation[1] of y under I. This definition is extremely general, and as any generalization, it is only interesting withing certain limits: although in principle any partial function could be considered an interpretation, we will use this term only for those functions that aim at capturing some meaning of the sentences to which it applies.

Although related, those notions have important differences. An interpretation function is by definition unambiguous: it associates at most a *single* interpretation to each sentence of its domain language (or none at all for some sentences, as it may be a partial function). On the other hand, we have seen that an sentence can have several meanings, depending on the agent interpreting it and their context, and many of those meanings can only be reached through multiple levels of interpretations. Those differences account for the ambiguity of the term “semantics”, used to denote meaning or interpretation depending on the context.

6.2. Syntax and semantics

Since the seminal works of Chomsky (1957), it is customary to define a formal language through its syntax and its semantics. Syntax is meant to discriminate, among a set of possible sentences, those that belong to the language being defined. Those valid sentences will then be interpreted thanks to the semantics. In other words, syntax focuses on the form, while semantics focuses on the content. Therefore it seems that syntax and semantics, while being intimately linked, are orthogonal to each other. But things are not as clean-cut.

Languages with no semantics?

XML (Bray et al. 1998, 2008) is a recommendation aiming to provide an interoperable syntax for exchanging digital documents, without presuming of the meaning ascribed to these documents, nor of their internal representation in the programs exchanging them. This intended agnosticism is the reason why the specification only addresses syntactic aspects (*i.e.* how to decide whether an XML document is well-formed[2] or not). This has lead many people to consider that XML was purely a syntax, with no semantics.

This is however an over-simplification. The syntactic constraints imposed by XML would be pointless if they didn’t allow a common interpretation of XML documents. That confusion can be attributed to two facts. First, this common interpretation exists but it is described in a separate recommendation, namely the Document Object Model (DOM) (Lauren Wood 1998), giving the impression that it is not an essential part of XML. Second, the DOM recommendation describes only *indirectly* the standard interpretation of XML documents: with the aim to stay neutral with respect to implementations, it does not describe the content of an XML document as a data structure, but as an abstract API allowing to programmatically interact with the document and its components.

However respectable that goal of neutrality, this choice was not suitable for many further specifications based on XML, which required more declarative descriptions of the structure of XML documents. Different such descriptions were therefore proposed (Clark and DeRose 1999; Cowan and Tobin 2004; Fernández et al. 2007), each of them based on a reading “between the lines” of the XML and DOM specifications, but each resulting to a formal model slightly different from the others.

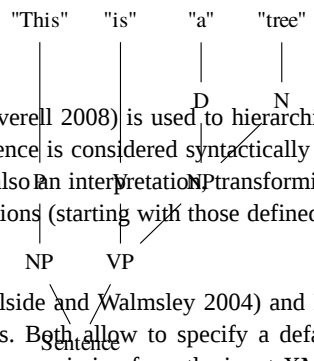
We can see here how the ambiguity of the term “semantics” lead to some confusion. It would have been better to acknowledge from the start that XML does have a syntax and a semantics, and describe the latter (the DOM tree) more explicitly. Still, it could have been emphasized that this first level of interpretation didn’t impose any in-memory representation, nor did it preclude any further interpretation of the DOM tree itself. It may also have prevented this confusion to happen again, as has been the case with JSON (Crockford 2006), successor of XML in some respects. JSON is said now and then to have no semantics, for the same reasons that motivated that claim about XML. Conversely, languages with an explicit formal semantics (typically knowledge representation languages, such as RDF) are expected by some to have some inherent advantage over so-called semantic-less languages, that would allow them to “magically” capture the whole meaning intended by an author.

Syntax and interpretation

Furthermore, syntax and semantics are not always as orthogonal as it seems.

First, for languages based on character strings, syntactic analysis is often split in two phases. The first one (lexical analysis) consists in grouping the characters into bigger units (tokens) that can be identified with simple rules (*e.g.* a sequence of letters

and digits), and dropping other irrelevant characters (*e.g.* spaces and punctuation marks). It is therefore a first interpretation, transforming a character string into a sequence of tokens.



In the second phase, a generative grammar (Chomsky 1957; Crocker and Overell 2008) is used to hierarchically decompose that sequence, according to a number of rules. If this process fails, the sequence is considered syntactically invalid. It it succeeds, the result is a parse tree (as the one in Fig. 6.1). Hence that phase is also an interpretation, transforming a sequence of tokens into a labelled tree, whose structure will be used by further interpretations (starting with those defined by the language semantics).

Some grammars go even further in interpreting the data. XML-Schema (Fallside and Walmsley 2004) and Relax-NG (Clark 2002) are two standards for specifying grammars of XML-based languages. Both allow to specify a default value for attributes. This means that, in the end of the syntactic analysis, if the attribute was missing from the input XML tree, it will be considered as present and holding the default value. In other word, the syntactic analysis transforms a possibly incomplete sentence into a complete one.

With those examples, we see that what is called syntax is often much more than a simple binary criterion for distinguishing valid sentences from invalid ones. It is instead a first chain of interpretations. Conversely, any interpretation I on a language straightforwardly induces a sub-language, namely the set of sentences interpretable under I (its domain of definition).

Relation with model theory

Model theory (Hodges 2013) is the mathematical foundation on which the semantics of many knowledge representation languages, including first-order logic, is defined. It is based on a notion called “interpretation”, which is different from the notion of the same name we have defined above. To avoid confusion, we name **MT-interpretation**s the interpretations defined by model theory.

More precisely, an MT-interpretation of a language L is a function I that maps the terms of L to the elements of a set Δ_I, and assigns a truth value to the sentences of L. We say that I **satisfies** a sentence s, or that I is a **model** of s, if and only if s is considered true under I.

The semantics of a language is not defined by a specific interpretation, but by a set of rules constraining which MT-interpretations are relevant for that language. The semantic properties of a sentence s are therefore defined by the set of *all* its models: s is *satisfiable* if it has at least one model; s is a *tautology* if it is true under every possible interpretation; s is a *consequence* of another sentence s' if every model of s' is also a model of s.

> **Example:** Let us consider a language where terms are lower case letters, upper case letters, and the character `=`. Sentences are sequences of those characters.
>
> Every MT-interpretation of that language maps:
>
> - to each lower case letter, an integer,
> - to each upper case letter, one of the operators +, -, × and /,
>
> and satisfies a sentence if and only if, by replacing the letters with their interpretations, one gets an arithmetic expression which is both correct and true.
>
> For example, the sentence s_1: $xAy = yAx = xBx$ has an infinite number of models, among which:
>
> - $\{x \to 3,\ \ y \to 2,\ \ A \to \times,\ \ B \to +\}$
> - $\{x \to 3,\ \ y \to 0,\ \ A \to \times,\ \ B \to -\}$
> - $\{x \to 1,\ \ y \to 0,\ \ A \to +,\ \ B \to \times\}$
>
> The sentence s_2: $xAy = xBx$ is satisfied by all models of s_1 above, it is therefore a consequence of s_1. Note that the opposite is not true, since
>
> - $\{x \to 1,\ \ y \to 0,\ \ A \to -,\ \ B \to \times\}$
>
> is a model of s_2 but not of s_1.
>
> **NB:** for the sake of simplicity, we have only considered MT-interpretation whose domain was the set of integers. A more realistic example would have allowed MT-interpretations to have any domain Δ_I. In that case, the constraints on MT-interpretation would have been to map

- to each lower case letter, an element of Δ_I,
- to each upper case letter, a function $f : \Delta_I \times \Delta_I \to \Delta_I$,

and the condition for satisfying a sentence would have to be rephrased in a more general fashion.

In a way, model theory acknowledges ambivalence, as it allows multiple MT-interpretations of the same language to coexist. This makes languages defined that way very versatile, as they are not restricted to a single interpretation, not even to a single interpretation domain. The drawback is that, by refusing to favor one particular model over the others, model theory can only recognized what is true in all of them. Somehow, it conflates all the models into a single interpretation, which can be seen as their "greatest common divisor". As a consequence, model theory is very likely to lose a part of the intended meaning of a language, and therefore should not be considered as the ultimate step in the interpretation process.

Summary

We have seen that the opposition between syntax and semantics is not a fruitful one, and that our notion of interpretation may provide a unified way to consider them. The multiple interpretations / representations of a sentence are linked together by interpretation functions defined at different levels. As an illustration, Fig. 6.2 shows the different languages and interpretation functions involved in interpreting an OWL ontology. Recall also that the meaning of a sentence is never unique, and that most language have several possible interpretations, which only the context allows us to chose. Fig. 6.3 gives an overview of this multiplicity through a few examples.

Fig. 6.2 Interpretation chain (nodes represent languages, edges represent interpretations)

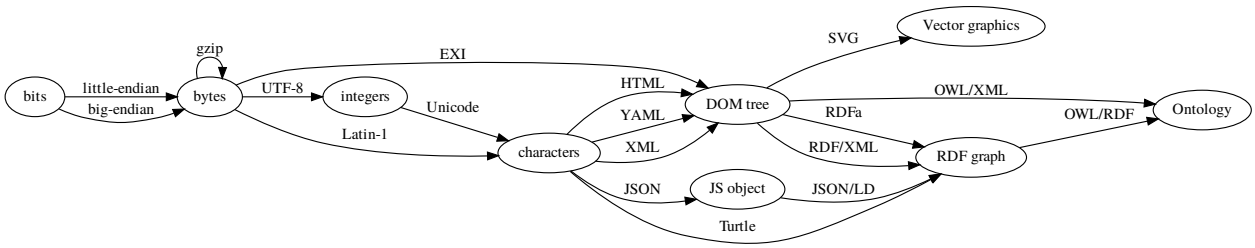


Fig. 6.3 Interpretation graph (nodes represent languages, edges represent interpretations)

6.3. Congruence

As mentioned above, although any partial function from one language to another satisfies our definition of interpretation, this notion is only relevant for some of those functions. We propose that a function can be considered as an interpretation on a language as soon as it accounts for some transformation or processing performed on the sentences of this language, by relating it to a transformation or processing on the interpreted sentences. In order to capture this intuition, we need a formal description on how those transformations are effectively related by interpretations.

Notations

As we are considering *partial* functions $f : L \to L'$, we need notations for denoting the domain of definition and the range of such functions:

$$\begin{aligned} In(f) &\stackrel{\text{def}}{=} \{x \in L |\ \exists y \in L', f(x) = y\} \\ Out(f) &\stackrel{\text{def}}{=} \{y \in L' |\ \exists x \in L, f(x) = y\} \end{aligned} \tag{1}$$

Definition

Let us consider two interpretation functions $I_1 : L_1 \to L'_1$ et $I_2 : L_2 \to L'_2$. The languages L_1 and L_2 constitute the realm of representations, whereas the languages L'_1 and L'_2 constitute the realm of interpretations. The notion of congruence

aims at capturing the fact that a function $f : L_1 \to L_2$ transforms representations *in accordance* with how a function $f' : L'_1 \to L'_2$ transforms their interpretations.

For this, we define the notions of **soundness** and **completeness**[3]. Intuitively, f is sound with respect to f' under (I_1, I_2) if every interpretable sentence computed by f corresponds to a sentence computed by f'. Conversely, f is complete with respect to f' under (I_1, I_2) if for every sentence (whose interpretation is) transformed by f', f computes the corresponding sentence. Formally:

$$sound_{I_1,I_2}(f, f') \iff In(I_2 \circ f) \subseteq In(f' \circ I_1) \wedge \forall x \in In(I_2 \circ f),\ \ I_2(f(x)) = f'(I_1(x)) \tag{2}$$

$$complete_{I_1,I_2}(f, f') \iff In(f' \circ I_1) \subseteq In(I_2 \circ f) \wedge \forall x \in In(f' \circ I_1),\ \ f'(I_1(x)) = I_2(f(x)) \tag{3}$$

Soundness and completeness are two forms of **congruence**, which we qualify as weak. When f is both sound and complete with respect to f' under (I_1, I_2), we say that f is **strongly congruent** to f' under (I_1, I_2). This conveys the idea that applying f to a representation amounts to apply f' to the corresponding interpretation. Fig. 6.4 proposes visual representations for soundness, completeness, strong congruence and unspecified congruence.

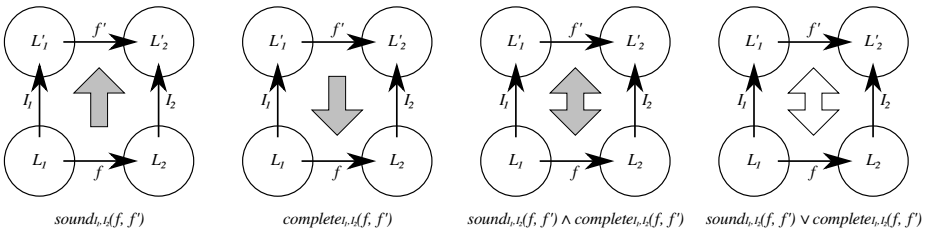


Fig. 6.4 Visual representation of congruence relations

Illustration

We illustrate here on an example the notions of congruence defined above. Let us consider the following languages and functions:

- $L_1 = L_2$ is the language of unicode strings of 10 characters or less $\mathbb{U}^{10}$;
- L'_1 is the set of natural numbers $\mathbb{N}$;
- L'_2 is the language of all sequences of natural numbers $\mathbb{N}^*$;
- $I_1 : \mathbb{U}^{10} \to \mathbb{N}$ interprets strings containing only digits as decimal representations of integers (*e.g.* “42”), even those containing spurious zeros (*e.g.* “042”);
- $I_2 : \mathbb{U}^{10} \to \mathbb{N}^*$ interprets strings containing only digits and spaces as sequences of integers, where spaces separate the items of the sequence, and digits are interpreted as in I_1;
- $f' : \mathbb{N} \to \mathbb{N}^*$ is the function transforming any positive natural number into the ordered sequence of its prime divisors (without repetition). For example, $f'(10) = (2, 5)$ and $f'(12) = f'(18) = (2, 3)$.

Notice that the definitions of congruence makes no hypothesis about the four functions I_1, I_2, f and f'. In particular, the functions defined above are

- not total (*i.e.* partial): “hello” has no interpretation under I_1 or I_2, 0 has no image under f';
- not injective: “42” and “042” have the same images under I_1 and I_2, $f'(12) = f'(18) = (2, 3)$;
- not surjective: 10^{10} has no representation under I_1 as strings in $\mathbb{U}^{10}$ are limited to 10 characters, $(2, 3, 5, 7, 11, 13)$ has no representation under I_2 for the same reason, $(5, 2)$ is not an image of f' since f' produces *ordered* sequences.

We will now study what it means for a function $f : \mathbb{U}^{10} \to \mathbb{U}^{10}$ to be congruent with f' under (I_1, I_2).

In order for f to be *sound*, every interpretable sentence it computes must correspond to a sentence computed by f'. So we could not have, for example, $f(\text{“}12\text{”}) = \text{“}2\ 5\text{”}$ or $f(\text{“}12\text{”}) = \text{“}3\ 2\text{”}$ else f would not be sound; instead we must have $f(\text{“}12\text{”}) = \text{“}2\ 3\text{”}$. By definition, the sentences for which f produces no output have no impact on soundness, so it is acceptable, for example, if $f(\text{“}4294967296\text{”})$ is undefined[4], even though $f'(4294967296)$ exists. Still by definition, the sentences for which f produces a non-interpretable output have no impact either on soundness, so it is equally acceptable, for example, if $f(\text{“}4294967296\text{”}) = \text{“}\mathrm{too\ big}\text{”}$. It follows that f could also be defined on non-interpretable sentences (for which

f' can not possibly have a corresponding output), as long as its output is also non-interpretable. For example, it is acceptable if $f(\text{“hello”})$ is undefined or returns “error”, but not if it returns “5”. Intuitively, this is because, in the latter case, the produced output could be mistaken for (the representation of) an output of f'.

In order for f to be *complete*, for every interpretable sentence transformed by f', f must compute the corresponding sentence. Therefore it is not acceptable anymore for $f(\text{“4294967296”})$to be undefined, it should return the correct value (“2”). Note that, on the other hand, it is now acceptable if $f(\text{“hello”})$ returns an interpretable sentence, such as “5”, as completeness is only concerned with interpretable inputs. For the same reason, the fact that $f'(10^{10}) = (2,5) = I_2(\text{“2 5”})$is not reflected by f, even though the *result* is representable, does not prevent f from being complete (nor sound). Intuitively, the notions of congruence are relative to the interpretations, and the fact that some inputs of f' have no representation under those interpretations should not be “held against” f for assessing its congruence.

Now, let us consider the case of $f'(30030) = (2,3,5,7,11,13)$ The input sentence is representable, but the output is not (its representation would not fit the 10 characters limit). In order to be sound, f must be undefined on “30030”, or return a non-interpretable sentence (*e.g.* “too long”). On the other hand, this case prevents any function $f : \mathbb{U}^{10} \to \mathbb{U}^{10}$ to be complete with respect to f' under I_1, I_2. Completeness could however be achieved by extending L_2 to accept longer strings.

Congruent predicates and relations

The notions of congruence we have just defined for functions can easily be extended to unary predicates and relations.

Considering two languages L and L', an interpretation function $I : L \to L'$, and two predicates $P \subseteq L$ et $P' \subseteq L'$, we define congruence relations as :

$$\begin{aligned} sound_I(P, P') &\iff \forall x \in L,\ \ P(x) \to x \in In(I) \wedge P'(I(x)) \\ complete_I(P, P') &\iff \forall x \in L,\ \ x \in In(I) \wedge P'(I(x)) \to P(x) \end{aligned} \tag{4}$$

Those definitions are of course closely related to definitions (2) et (3). In fact, we can replace predicate P with a function $f_P : L \to \{\top\}$ mapping $\top$ to all sentences of L verifying P, an only to them (and respectievely for L' and P'). The congruence of P to P' under I is then equivalent to the congruence of f_P to $f_{P'}$ under (I, id), where id is the identity function on $\{\top\}$.

Extending this to binary or n-ary relations is straightforward, as L and L' could be defined as the cartesian product of several sub-languages L_i and L'_i respectively, and I as the combination of several interpretation functions $I_i : L_i \to L'_i$.

In the special case where $L = L'$ and where the interpretation is the identity function, then those definitions can be simplified to:

$$\begin{aligned} sound(P, P') &\iff \forall x \in L,\ \ P(x) \to P'(x) \\ complete(P, P') &\iff \forall x \in L,\ \ P'(x) \to P(x) \end{aligned} \tag{5}$$

Properties

We present here a number of notable properties of congruence relations.

In the following, we consider languages $L_1, L_2, L_3, L'_1, L'_2, L'_3, L''_1, L''_2$, and functions

- $I_1 : L_1 \to L'_1$
- $I_2 : L_2 \to L'_2$
- $I_3 : L_3 \to L'_3$
- $I'_1 : L'_1 \to L''_1$
- $I'_2 : L'_2 \to L''_2$
- $f : L_1 \to L_2$
- $f' : L'_1 \to L'_2$
- $f'' : L''_1 \to L''_2$
- $g : L_2 \to L_3$
- $g' : L'_2 \to L'_3$
- $h : L_1 \to L_3$
- $h' : L'_1 \to L'_3$

Symmetry

If f is sound with respect to f' under (I_1, I_2), then I_1 is complete with respect to I_2 under (f, f'), and conversely.

$$sound_{I_1,I_2}(f, f') \leftrightarrow complete_{f,f'}(I_1, I_2) \tag{6}$$

In this equivalence, the transformation functions and the interpretation functions switch roles. As noted earlier, this may be relevant only with certain functions f and f' and in certain contexts.

Transivitivy

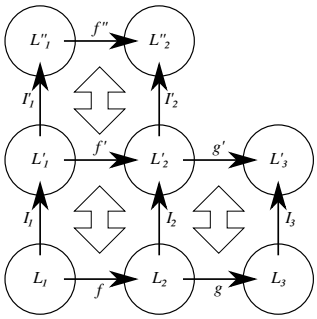


Fig. 6.5 Composition of congruence relations

Congruence properties are transitive through composition, either “horizontal” (*i.e.* applied to the congruent functions) or “vertical” (*i.e.* applied to the interpretation functions).

$$
\begin{aligned}
sound_{I_1,I_2}(f,f') \wedge sound_{I_2,I_3}(g,g') &\rightarrow sound_{I_1,I_3}(g \circ f, g' \circ f') \\
complete_{I_1,I_2}(f,f') \wedge complete_{I_2,I_3}(g,g') &\rightarrow complete_{I_1,I_3}(g \circ f, g' \circ f') \\
sound_{I_1,I_2}(f,f') \wedge sound_{I'_1,I'_2}(f',f'') &\rightarrow sound_{I'_1 \circ I_1, I'_2 \circ I_2}(f,f'') \\
complete_{I_1,I_2}(f,f') \wedge complete_{I'_1,I'_2}(f',f'') &\rightarrow complete_{I'_1 \circ I_1, I'_2 \circ I_2}(f,f'')
\end{aligned}
\tag{7}
$$

The “vertical” version of that property is of particular interest when considering interpretation chains (such as the one represented in Fig. 6.2).

Associativity

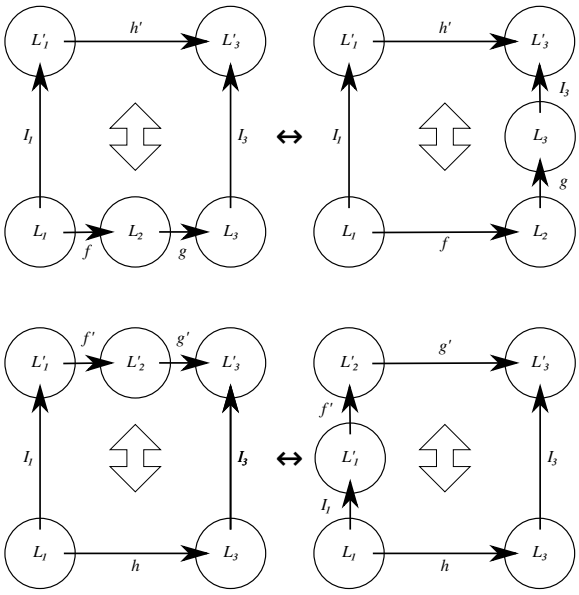


Fig. 6.6 Associativity of congruence relations

When one of the four functions involved in a congruence relation can be expressed as a function composition, it can be decomposed and recomposed with its “adjacent” function, while preserving the congruence properties.

(8)

$$
\begin{aligned}
sound_{I_1,I_3}(g \circ f, h') &\leftrightarrow sound_{I_1,I_3 \circ g}(f, h') \\
complete_{I_1,I_3}(g \circ f, h') &\leftrightarrow complete_{I_1,I_3 \circ g}(f, h') \\
sound_{I_1,I_3}(h, g' \circ f') &\leftrightarrow sound_{f' \circ I_1,I_3}(h, g') \\
complete_{I_1,I_3}(h, g' \circ f') &\leftrightarrow complete_{f' \circ I_1,I_3}(h, g')
\end{aligned}
$$

As with the property of symmetry above, one of the function (g and f' in the definitions above) changes role, from transformation to interpretation or conversely. This demonstrates how this distinction is relative, and depends on the point of view.

Let us examine a special case where this property applies: when one of the four function is the identity function id. Indeed, any function can be seen as the composition of itself with the identity function: $f = f \circ \mathrm{id} = \mathrm{id} \circ f$. The equations above can, in that case, be rewritten as:

$$
\begin{aligned}
sound_{I_1,I_2}(f, \mathrm{id}) &\leftrightarrow sound_{\mathrm{id},I_2}(f, I_1) \\
complete_{I_1,I_2}(f, \mathrm{id}) &\leftrightarrow complete_{\mathrm{id},I_2}(f, I_1) \\
sound_{I_1,I_2}(\mathrm{id}, f') &\leftrightarrow sound_{I_1,\mathrm{id}}(I_2, f') \\
complete_{I_1,I_2}(\mathrm{id}, f') &\leftrightarrow complete_{I_1,\mathrm{id}}(I_2, f')
\end{aligned}
\tag{9}
$$

Inverse interpretations

If I_1 and I_2 are invertible, then we might wonder how congruence under (I_1, I_2) affects congruence under (I_1^-, I_2^-). This happens only if I_1 and I_2 have certain properties on the domains of f and f', respectively:

$$
\begin{aligned}
sound_{I_1,I_2}(f, f') \wedge In(I_2) \supseteq Out(f) &\rightarrow complete_{I_1^-,I_2^-}(f', f) \\
complete_{I_1,I_2}(f, f') \wedge Out(I_1) \supseteq In(f') &\rightarrow sound_{I_1^-,I_2^-}(f', f)
\end{aligned}
\tag{10}
$$

In particular, those properties are verified if I_1 (respectively I_2) is a bijection between L_1 and L_1' (respectively between L_2 and L_2').

Equivalence relations

Here we consider two relations $R \subseteq L_1 \times L_1$ et $R' \subseteq L_1' \times L_1'$. We want to determine how congruence of R with respect to R' propagates the properties of an equivalence relation between R and R'.

It can be shown that, if R is strongly congruent to R' under $I_1 \times I_1$, then R is reflexive (respectively symmetric, transitive) on L_1 if and only if R' is reflexive (respectively symmetric, transitive) on L_1' [5].

Note that for every function $f : L_1 \rightarrow L_2$, we can define the relation $\equiv_f$ on $In(f)$ as :

$$
\forall x, y \in In(f), \quad x \equiv_f y \iff f(x) = f(y) \tag{11}
$$

By definition, this relation is strongly congruent with equality under $f \times f$, which satisfies the intuition between the notion of "equivalence relation": x and y are equivalent as they can be *interpreted as equal* (according to f).

Connection with mathematical logic

It has probably not eluded the logically inclined reader that the terms "sound" and "complete" are borrowed from mathematical logic. This is because the notions of soundness and completeness in this field are a special case of the notions proposed here: consider L the set of valid formulae of a formal system S, P the predicate $\vdash$ indicating that a sentence is derivable in S, and P' the predicate $\vDash$ indicating that a sentence is a tautology. Then the equations (5) above coincide with the classical notions of soundness and completeness, becoming:

$$
\begin{aligned}
sound(\vdash, \vDash) &\iff \forall s \in L, \quad \vdash s \rightarrow \vDash s \\
complete(\vdash, \vDash) &\iff \forall s \in L, \quad \vDash s \rightarrow \vdash s
\end{aligned}
\tag{12}
$$

Following the steps of Hofstadter (1979), we can also rephrase Gödel's first incompleteness theorem in our framework[6]. For any (sufficiently expressive) formal system S on a language L, there exist:

- an unambiguous (invertible) representation of every sentence s of L by a natural number $G(s)$ – or conversely, an interpretation function $G^- : \mathbb{N} \rightarrow L$;
- a certain computation C such that, for any sentence s, $C(G(s))$ verifies that s is derivable in S – in other words, C

(considered as a predicate on $\mathbb{N}$) is strongly congruent with $\vdash$ under G^-;
- a number γ such that the sentence $\neg C(\gamma)$ is represented by γ itself through G.

The last point is the cornerstone of Gödel's proof: the sentence $G^-(\gamma)$ states "γ does not satisfy C", which can also be interpreted as "$G^-(\gamma)$ can not be derived". Unless the system was inconsistent, neither the sentence or its negation can be derived in that formal system (Raatikainen 2015), it is undecidable.

The ambivalence is key in this famous result: it uses the fact that the sentence $G^-(\gamma)$, known as the Gödel sentence of the system S, can be read at two different levels (a statement about the number γ, and a statement about itself). But, as Hofstadter points out, it would be naive to think of any of these statements as the correct or final interpretation of that sentence. Although one may be tempted to get rid of the undecidability by adding the Gödel sentence as an axiom, there will still be infinitely many interpretations on L, and only one of them would be "cured" in the process. In other words, there would still be another sentence in L which, according to another interpretation, would be a Gödel sentence for the new system.

6.4. Ambivalence

This notion of congruence provides us with a theoretical and formal framework for apprehending ambivalence. Considering that any functionality of a computer system can be reduced to a function transforming data, then any interpretation of those data allowing to justify or explain that transformation (in terms of congruence to another transformation) can be deemed relevant, even if it differs from the original intent of the system.

Software development

In it simplest form, developing (a functionality of) a software application amounts to implement a function p (the program) that is congruent to a function s (the specification). The program works on digital representations, while the specification can be defined at an arbitrary level of abstraction. The interpretation functions allowing to cross those abstraction levels, and therefore to establish the congruence of p to s, depend on the development and execution environments. Ultimately, the computer handles bits, which are interpreted by the machine language as integers or floating point numbers. In C, those can be further interpreted as ASCII characters, or composed into more complex structures (arrays, structured types). Additional libraries may provide further interpretation layers: a geometry library may consider arrays as points or vectors; an XML library may consider character strings as DOM trees.

Note that the example interpretations listed above are of two kinds. Some of them are purely conventional, where the same data in memory is considered differently (*e.g.* bits, number, character) by different parts of the code. Others involve an actual rewriting of the data to support the interpretation: this is usually the case for XML libraries, where the DOM tree is "materialized" into a dedicated data structure through a parsing process. This latter example illustrates again how the distinction between transformation and interpretation is contextual.

It follows that, as soon as the specification is expressed beyond the abstraction level provided by the environment, the developer's work is not anymore reduced to implementing p. It also consists in inventing new interpretation functions justifying the desired congruence, either as new conventions, or as new data structures with their associated parsing functions. Those considerations are of course not new: software engineering methodologies have long identified the need to iteratively decompose an abstract specification in order to implement it. The object-oriented paradigm (Meyer 1997), in particular, emphasizes this approach. Furthermore, the most useful and reusable abstractions have been identified and formalized as design patterns (Gamma et al. 1995), or even integrated to programming languages (such as strongly object-oriented programming languages).

As valuable as this trend may have been, helping developers to reuse proven and largely understood abstractions, it also lead to a rigidification of practices, to which agile development methods can be seen as a reaction. Those methods endorse the fact that applications provide unplanned affordances, and end up being misused, adapted or diverted, in other words *interpreted* in multiple ways, different from the originally intended interpretation. By insisting on continuous delivery, they allow development teams to identify those alternative interpretations early on, and to adjust to them. Refactoring, another important component of agile development, can be seen as the process of changing data and program structures (as a result of altering interpretations) while preserving the congruence of the program with the intended specification.

Program and data

We have proposed above a point of view on refactoring where the program is consider as a function. Another way to look at it is to consider the program as a sentence (in a programming language) interpretable according to a given interpretation function: the standard semantics of the programming language, specifying the behaviour that each program must have. With this point of view, refactoring can be seen as a transformation of program-sentences, which has to be congruent with identity under the standard semantics (*i.e.* to preserve the behaviour).

This duality between program and data has long been recognized, even though the distinction is often posed as a working hypothesis for practical reasons. For example, in a Turing machine, the program (*i.e.* the set of rules specifying the behaviour of the machine) is “embedded” in the machine and static, while the data stored on the tape are modifiable. This distinction is also present in classical computer architecture, where the memory used by a process is divided into a code segment, usually read-only, containing the executable machine code of the program, and a writable data segment, containing the data handled by that program.

Still, the boundary between the two is relative. Turing (1936) proved the existence of a universal machine, able to simulate any other Turing machine described on its tape. Similarly, many programming languages nowadays are interpreted[7]. In those cases, the program segment only contains the interpreter’s code (a kind of universal machine) while the application program itself is stored in the data segment with its own data, suggesting a threefold partition (interpreter-program-data) instead of the initial twofold partition. Then some libraries also make use of so-called domain-specific languages, or mini-languages (Raymond 2003, chap. 8), promoting a part of the data to yet another level of “program-ness”[8].

In Artificial Intelligence, the classical program-data distinction has also been largely questioned, in search of better alternatives. Knowledge Based Systems can be seen as such an alternative, distinguishing a generic reasoning engine from a domain-specific knowledge-base. Further partitions of the knowledge base itself have also been proposed: rules and facts in Prolog, T-box and A-box in description logics (Baader et al. 2003)... We see that different users (*e.g.* system administrator, developer, knowledge engineer, end user) will have different points of view on which part of the information in the computer’s memory is being executed, and which part is being merely processed by the former.

Self-rewriting programs, and other kinds of meta-programming, muddy the waters a little more regarding the program-data distinction. Arsac (1988) testifies to such practices as soon as 1965. It is interesting to notice that he calls them “instruction computation”, which highlights the different abstraction levels at work: the semantics of the program (instructions with an intended behaviour) and its representation in memory (numbers produced by computation). It is even more interesting that, according to Arsac, some people contested that designation, arguing that not all computations were relevant for instructions – which could be restated in our framework: not all computations are congruent to a meaningful transformation of instructions.

Representations and intelligibility

The user of a computer system apprehends the underlying data through their perceivable representation(s) offered by the system. More precisely, the system internally processes sentences of a language L_i, and transforms them into a language L_p perceivable by the user (*e.g.* textual, graphical, audible). Both languages aim to represent conceptual structures targeted by the system, sentences of a language L_c. Therefore, there are interpretation functions $I_i : L_i \to L_c$ and $I_p : L_p \to L_p$, and the **presentation** function $p : L_i \to L_p$ must be strongly congruent to the identity function on L_c.

The requirement for p to be congruent to the identity, rather than to some kind of projection, may seem too strong. Indeed, complex structures are often better apprehended through several partial but complementary representations than through a single exhaustive one, as can be seen in the interface of Advene (see Fig. 5.1) or of 3D modelling applications (see Fig. 6.7). Furthermore, in many applications, the user is never presented with the whole information at once, but has access to it by browsing from one partial representation to another. We can however justify that constraint as follows. First, we consider a function p returning the whole set of partial representations available to the user (rather than those effectively rendered at a given time). Second, we consider that if any information was totally invisible to the user, including through their interactions with the system, then we can as well assume that this information is absent from the system.

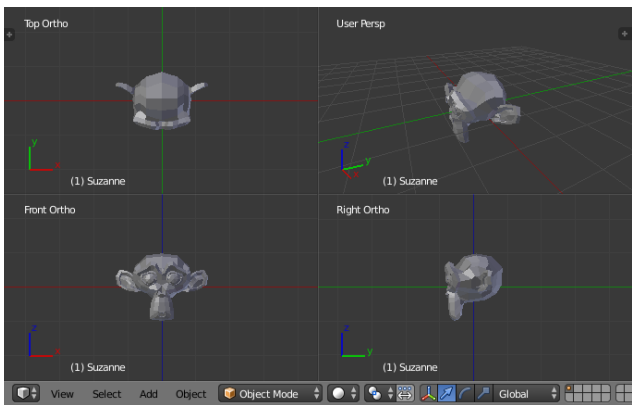


Fig. 6.7 Presentation of a 3D model through several projections (source: Blender Manual[9])

We can then remark that the user may interpret the representations available to them differently from the interpretation I_p, intended by the system designers, which we may call the **canonical interpretation**, as a reference to Prié (2011). Some social codes (typographical conventions, standard GUI patterns, *etc.*) may limit this divergence, but can rarely prevent it entirely. Prié also notices that presentation can induce in the user’s interpretation some structures that do not exist in the underlying data. For example, in a word processing application, the fact that two characters are vertically aligned in a paragraph is an effect of presentation, without any counterpart in the data. However, the user may integrate this in their interpretation, and exploit it in their practice (in order, for example, to create a graphical effect in the text). Similarly, the user apprehends the functionalities of the system through presentation. Every operation on the data reflects on their presentation, and if the perceived change is

congruent with the functionality assumed by the user, this will confirm their interpretation. On the other hand, a mismatch will lead them to revise their interpretation – or to consider that the system is not working properly[10]. Following our example above, the user may be frustrated that a copying and pasting a the paragraph on a page with a different width does not preserve the vertical alignment of its characters.

Therefore, there is a tension in the design of a presentation. Specific presentations help the user conceive an appropriate interpretation (*i.e.* close enough to the canonical interpretation), and provide meaningful affordances to the system's functionalities. Generic presentations, on the other hand, are less helpful at first, but allow the user to adapt the system to their own interpretations and use cases, even if these diverge slightly from the ones originally intended by the system designers.

The aim of this chapter was to try and elicit a number of intuitions that drove most of the works presented in this dissertation. The proposed formal framework offers an alternative approach to traditional notions of semantics. In this framework, interpretations are not considered only as a prerequisite to standard processings (although they can still be used that way), but can also become retrospective justification of *ad-hoc* processings. Semantics is therefore open-ended, potential interpretations being never exhausted. Work remains to be done, however, to make this framework operational, and provide effective guidelines to build ambivalence-aware systems.

Notes

[1] NB: if I is not injective, then the inverse relation is not itself a function, and y can have *several* representations under I.

[2] Actually, XML distinguishes two levels of compliance, well-formedness and validity, but both are syntactic criteria.

[3] Those terms are borrowed from mathematical logic; we will show in the end of this section that the definitions we give here are a generalization of their usual definitions.

[4] This could be the case if f was a computer program using 32-bits integers.

[5] In fact, it is enough if R (respectively R') has those properties on $In(I_1)$ (respectively $Out(I_1)$). They do not have to be verified on the whole of L_1 or L'_1.

[6] Note that this incompleteness is *not* the opposite of $complete(\vdash, \models)$, but of a subtly different notion of completeness. Namely, a formal system S on a language L is complete if every sentence of L or its negation can be derived in S. In other words, if S is incomplete in that sense, some sentences do not have the same truth value in all possible models.

[7] This includes languages such as Java and C#. Although those languages require a compilation phase, they are not compiled into native machine code, but to a lower-level language (bytecode) that still needs an external native program to be executed. So strictly speaking, this lower-level language is an interpreted language.

[8] This is humourously summarized by Greenspun's Tenth Rule of Programming: "Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified bug-ridden slow implementation of half of Common Lisp" (http://philip.greenspun.com/research/). Greenspun's intent here is cleary to encourage to use high-level languages (such as Common Lisp) instead of re-implementing their functionalities in C or Fortran. But this aphorism also suggests that one could consider a part of the low-level program as an interpreter, and the other part as a higher-level program interpreted by the former.

[9] https://www.blender.org/manual/editors/3dview/navigate/3d_view.html

[10] In which case they might be opposed the famous meme "it is not a bug, it is a feature", which we could rephrase: "if the system is not congruent under your interpretation, then your interpretation is wrong, not the system".

Chapter bibliography

Arsac, Jacques. 1988. "Des Ordinateurs à l'informatique (1952-1972)." In , edited by Philippe Chatelin, 1:31–43. Grenoble. http://jacques-andre.fr/chi/chi88/arsac.html.

Baader, Franz, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Bray, Tim, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. 1998. "Extensible Markup Language (XML) 1.0." W3C Recommendation. W3C. http://www.w3.org/TR/1998/REC-xml-19980210.

———. 2008. "Extensible Markup Language (XML) 1.0 (Fifth Edition)." W3C Recommendation. W3C. http://www.w3.org/TR/xml/.

Chomsky, Noam. 1957. *Syntactic Structures*. Mouton.

Clark, James. 2002. "RELAX NG Compact Syntax." OASIS Committee Specification. OASIS. http://www.oasis-open.org/committees/relax-ng/compact-20021121.html.

Clark, James, and Steve DeRose. 1999. “XML Path Language (XPath).” W3C Recommendation. W3C. http://www.w3.org/TR/xpath/.

Cowan, John, and Richard Tobin. 2004. “XML Information Set (Second Edition).” W3C Recommendation. W3C. http://www.w3.org/TR/xml-infoset/.

Crocker, David, and Paul Overell. 2008. “Augmented BNF for Syntax Specifications: ABNF.” RFC 5234. IETF. ftp://ftp.rfc-editor.org/in-notes/std/std68.txt.

Crockford, Douglas. 2006. “The Application/Json Media Type for JavaScript Object Notation (JSON).” RFC 4627. IETF. http://www.ietf.org/rfc/rfc4627.txt.

Fallside, David C., and Priscilla Walmsley. 2004. “XML Schema Part 0: Primer Second Edition.” W3C Recommendation. W3C. http://www.w3.org/TR/xmlschema-0/.

Fernández, Mary, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. 2007. “XQuery 1.0 and XPath 2.0 Data Model (XDM).” W3C Recommendation. W3C. http://www.w3.org/TR/xpath-datamodel/.

Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley.

Hodges, Wilfrid. 2013. “Model Theory.” In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Fall 2013. http://plato.stanford.edu/archives/fall2013/entries/model-theory/.

Hofstadter, Douglas R. 1979. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Basic Books.

Lauren Wood. 1998. “Document Object Model (DOM) Level 1 Specification.” W3C Recommendation. W3C. http://www.w3.org/TR/REC-DOM-Level-1/.

Meyer, Bertrand. 1997. *Object-Oriented Software Construction, Second Edition*. Prentice Hall.

Nivat, Maurice, and Andreas Podelski. 1992. *Tree Automata and Languages*. Amsterdam; New York: North-Hollandi. http://books.google.com/books?id=x91QAAAAMAAJ.

Pemberton, Steven. 2000. “XHTML 1.0 The Extensible HyperText Markup Language (Second Edition).” W3C Recommendation. W3C. https://www.w3.org/TR/xhtml1/.

Prié, Yannick. 2011. “Vers une phénoménologie des inscriptions numériques. Dynamique de l’activité et des structures informationnelles dans les systèmes d’interprétation.” Thesis, Université Claude Bernard - Lyon I. https://tel.archives-ouvertes.fr/tel-00655574/document.

Raatikainen, Panu. 2015. “Gödel’s Incompleteness Theorems.” In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Spring 2015. http://plato.stanford.edu/archives/spr2015/entries/goedel-incompleteness/.

Raymond, E. S. 2003. *The Art of Unix Programming*. Pearson Education.

Rozenberg, Grzegorz, ed. 1997. *Handbook of Graph Grammars and Computing by Graph Transformation*. Singapore ; New Jersey: World Scientific.

Schreiber, Guss, and Yves Raimond. 2014. “RDF 1.1 Primer.” W3C Working Group Note. W3C. http://www.w3.org/TR/rdf-primer/.

The Unicode Consortium. 2016. *The Unicode Standard, Version 9.0.0*. Mountain View, CA: The Unicode Consortium. http://www.unicode.org/versions/Unicode9.0.0/.

Turing, Alan M. 1936. “On Computable Numbers, with an Application to the Entscheidungsproblem.” *J. of Math* 58 (345–363): 5.

7. Perspectives and conclusion

To conclude this dissertation, I will propose a number of directions in which the works presented therein can be improved, continued and extended.

7.1. Convergences

Several of the works presented in this dissertations have pioneered the use of Web technologies in fields were they were not considered at the time. The modelling of interaction traces with RDF allowed us to elicit the semantics of the constituents of the traces, and to share reusable trace models and transformations. As for video annotations, we experimented with HTML-based hypervideos before video became a first-class citizen of the Web. In the meantime, those use-cases have gained traction, and W3C standards have emerged to support them. It is therefore necessary to re-evaluate our original proposals in the light of those standards, the evolution of technologies, and the evolution of uses.

As mentioned in the end of Section 4.1, we have already started this critical work (Cazenave-Lévêque 2016) comparing our modelled-trace meta-model in particular to PROV (Lebo, Satya Sahoo, and McGuinness 2013) and Activity Streams (Snell and Prodromou 2016). While each has its own focus, they both address the problem of representing some agents' activity. More work is required to align our meta-model with those standards, especially when it comes to capturing the *context* of the activity. However, once this is done, we could leverage PROV traces and Activity Streams using Trace Based Reasoning (see Section 2.4). Since both formats are being increasingly adopted for representing traces in in various domains, but provide no standard mean for processing or tapping these traces, this will create a number of stimulating opportunities for applying and refining TBR.

The same goes for video annotations: the Web Annotation Data Model (Sanderson, Ciccarese, and Young 2016), combined with Media Fragment URIs (Troncy et al. 2012), now provides a standard replacement for Cinelab annotations (Section 5.1). However, the Cinelab model goes beyond the scope of Web Annotations, proposing to describe and share reusable annotation structures (annotation models), as well as hypervideos based on those annotations (views). We plan to adapt those concepts to the Web Annotation standard. In particular, although a number of popular Javascript toolkits have been proposed to produce rich HTML5-based hypervideos, a more declarative approach would ease the authoring and reuse of hypervideo designs. Our proposals in that direction based on Cinelab (Sadallah, Aubert, and Prié 2014; Steiner et al. 2015) ought to be adapted to the new Web Annotation model.

7.2. Towards ambivalence-aware intelligent applications

Winograd (2006) suggests that the perceived opposition between AI and Human Computer Interaction (HCI) is actually a deeper opposition between what he calls the rationalistic approach (putting emphasis on data and knowledge processing) and the design approach (putting emphasis on interactions). He argues that a balance must be found between those two trends, both in AI and in HCI, and that both fields may not differ so much once the right balance is found.

I believe that the proposal in Chapter 6 may be helpful in finding that balance. In the end of of Section 6.4, we have emphasized the tight relationship between semantics and interactions, and proposed that the notion of congruence can help design interfaces and visualizations supporting multiple interpretations. This will of course require to investigate more deeply the implications of the proposed framework. At a theoretical level, we must precisely define the desirable formal properties that the notion of congruence can bring into a system. This will then allow us, at a practical level, to provide guidelines for designing ambivalence-aware systems and user interfaces, and experimentally assess their added-value compared to more traditional systems.

More recently, another opposition has been dividing AI, which we could label top-down versus bottom-up. The former ap-

proach focuses on knowledge engineering, and interpreting data through carefully designed models, while the latter focuses on data mining and machine learning, leaving semantics to emerge from the data. Elated by the spectacular successes of deep-learning techniques (Johnson 2016; Williams 2016), some people have pronounced the definitive victory of the bootom-up approach. Wu (2013) claims that “having more data allows the ‘data to speak for itself’, instead of relying on unproven assumptions and weak correlations”. This extreme position is neglecting the biases introduced by how data is collected, represented, visualized and more generally, made to “speak”. Surely, both approaches are valuable and can complement each other as long as they leave room for ambivalence, rather than aiming for an illusory unique objective meaning.

Existing approaches have been successful for big organizations and companies, having access to massive amounts of data generated by their users, and to the pertaining computing power required to process them. Novel approaches are still required, however, to help individuals make sense of their own personal data. These data are often too sparse for pure bottom-up approaches, and too heterogeneous for pure top-down approaches. They are too complex for manual processing (even with efficient user interfaces), but also too complex for fully automated reasoning. Again, the proposal in Chapter 6 can provide the basis of a unifying framework for such hybrid approaches.

7.3. Re-decentralizing the Web

Regardless of the approach, users willing to analyze their personal data face another problem: a large part of these data is locked in the databases of the services they use, and they only have a limited access to them, if any. What is worse, users have very little control or knowledge on who *has* access to their personal data.

As stressed by Hall, Hendler, and Staab (2016), the Web has become so influent in our societies that it can no more be considered only as a technical system. “Code becomes law, but the law should not be imposed by the few without the control, or at least the knowledge, of the many.” Any technical decision must be considered with its social and ethical implications. We must therefore favor architectures empowering users. In Chapter 4, we have shown how the principles of Linked Data can help achieve this goal. Projects such as SOLID (Mansour et al. 2016) and Hydra (Lanthaler and Gütl 2013) allow to create Web services which do not deprive users from control over their personal data, and which can connect with each other on the basis of users needs and intents, rather than on a pre-defined application-driven basis. All the future works identified in this conclusion must keep in line with this trend, and contribute to it.

7.4. Conclusion

When starting to write a habilitation dissertation, it is at first difficult to find a consistent way to present one’s work. It looks very much as a patchwork, each project involving different people, happening in different contexts, and led much more by opportunities and serendipity than by a continuous attempt to hold one’s course. Is there any consistency there to find anyway? One then has to adopt a different perspective on the work done, focusing on the only common trait in all this diversity: oneself. Then one can try and describe not only what they have done, but why they did it that way, and what remains to be done.

Writing this dissertation has been a long undertaking. It was nevertheless a rewarding experience: first by helping me realize that I was indeed following a more consistent research direction that I would have thought; and second by giving me an opportunity to rediscover my own works (and others’) in the light of this research direction. I hope that the reader will have found as much interest in reading these pages that I have found in writing them.

As I tried to demonstrate along those chapters, what relates our works on interaction traces, on video annotations and on the Web of linked data, is the ambition to build intelligent systems leaving the field of possible interpretations as open as possible. That way, the system can adapt to the user, rather than forcing the user to adapt to it. The ability for users to define their own transformations in MTMSs, or their own description schemas and views in Advene, enables them to elicit and leverage their own interpretations.

It does not mean, however, that all possible interpretations are equally valid; meaning can not be arbitrarily decreed (despite Humpty Dumpty’s attempt, see the excerpt opposite), it has to be co-constructed, negotiated. The notion of congruence, proposed in the last chapter, aims to provide a formal framework to account for this negotiation: an interpretation is acceptable to the extent that it explains the operations that are performed (or can be performed) with the system.

Semantics is therefore anchored in interactions, which brings us back to the importance of traces and TBR, of course, but also to the importance of design. In order to be acknowledged as intelligent, a system must first be intelligible. In Section 6.4, we have explored the potential impact of the proposed framework on different aspects of software design, including intelligibility of information presentation. By describing a formal link between interpretations, on the one hand, and presentations and transformations, on the other hand, (in other words, between what the system means, and what it does), we may contribute to find a balance in the “rationalistic/design” opposition identified by Winograd (2006).

Finally, the role of the Web in the design of modern intelligent systems can not be neglected. The Web is not a separate appli-

cation domain, it is the backdrop to all user's interactions with any system or artefact, even non-connected ones, even non-digital ones: people will search forums for help about an application, they will google an unknown word from a book, they will not be surprised to see a URL printed on the side of a bus, even if this "link" is not directly actionable. It can be argued that, nowadays, there is no such thing as a non-connected object.

The value of the Web is not only in the amount of information available to us, but also (and mostly) in the numerous links that interconnect this information. Every link puts a piece of information in the context of many others, allowing as many interpretations and re-interpretations, provided that we have the right tools to take advantage of this ambivalence. This is the kind of tools that Bush (1945) had in mind when he imagined the Memex. Instead, as Pariser (2011) warns us, many current tools collapse this multiplicity, providing each of us with a comfortable, personalized and unique perspective, each to his own. We must therefore make sure that the Web's ambivalence is not reduced to a juxtaposition of idiosyncrasies, but remains an empowering network of intertwined trails.

The question of meaning

`I don't know what you mean by "glory,"' Alice said.

Humpty Dumpty smiled contemptuously. `Of course you don't – till I tell you. I meant "there's a nice knock-down argument for you!"'

`But "glory" doesn't mean "a nice knock-down argument,"' Alice objected.

`When I use a word,' Humpty Dumpty said in rather a scornful tone, `it means just what I choose it to mean – neither more nor less.'

`The question is,' said Alice, `whether you can make words mean so many different things.'

`The question is,' said Humpty Dumpty, `which is to be master – that's all.'

– Lewis Carroll (1871)

Chapter bibliography

Bush, Vannevar. 1945. "As We May Think." *The Atlantic Monthly*, July. http://www.liacs.nl/~fverbeek/courses/hci/memex-vbush.pdf.

Carroll, Lewis. 1871. *Through the Looking-Glass*. http://www.gutenberg.org/ebooks/12.

Cazenave-Lévêque, Raphaël. 2016. "Interoperability among Trace Formalisms." Master's Thesis, Université Lyon 1.

Hall, Wendy, Jim Hendler, and Steffen Staab. 2016. "Web Science Manifesto." The Web Science Trust. November 22, 2016. http://www.webscience.org/manifesto/.

Johnson, George. 2016. "To Beat Go Champion, Google's Program Needed a Human Army." *The New York Times*, April 4, 2016. http://www.nytimes.com/2016/04/05/science/google-alphago-artificial-intelligence.html.

Lanthaler, Markus, and Christian Gütl. 2013. "Model Your Application Domain, Not Your JSON Structures." In *Proceedings of the 22Nd International Conference on World Wide Web*, 1415–1420. WWW '13 Companion. New York, NY, USA: ACM. https://doi.org/10.1145/2487788.2488184.

Lebo, Timothy, Satya Sahoo, and Deborah McGuinness. 2013. "PROV-O: The PROV Ontology." W3C Recommendation. W3C. https://www.w3.org/TR/prov-o/.

Mansour, Essam, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. "A Demonstration of the Solid Platform for Social Web Applications." In *Proceedings of the 25th International Conference Companion on World Wide Web*, 223–226. International World Wide Web Conferences Steering Committee. http://dl.acm.org/citation.cfm?id=2890529.

Pariser, Eli. 2011. *Beware Online "Filter Bubbles."* TED Talk. https://www.ted.com/talks/eli_pariser_beware_online_filter_bubbles.

Sadallah, Madjid, Olivier Aubert, and Yannick Prié. 2014. "CHM: An Annotation- and Component-Based Hypervideo Model for the Web." *Multimedia Tools and Applications* 70 (2): 869–903. https://doi.org/10.1007/s11042-012-1177-y.

Sanderson, Robert, Paolo Ciccarese, and Benjamin Young. 2016. "Web Annotation Data Model." W3C Candidate Recommendation. W3C. https://www.w3.org/TR/annotation-model/.

Snell, James, and Evan Prodromou. 2016. "Activity Streams 2.0." W3C Candidate Recommendation. W3C. http://www.w3.org/TR/activitystreams-core/.

Steiner, Thomas, Rémi Ronfard, Pierre-Antoine Champin, Benoît Encelle, and Yannick Prié. 2015. "Curtains Up! Lights, Camera, Action! Documenting the Creation of Theater and Opera Productions with Linked Data and Web Technologies." In . https://hal.inria.fr/hal-01159826/document.

Troncy, Raphaël, Erik Mannens, Silvia Pfeiffer, and Davy Van Deursen. 2012. "Media Fragments URI 1.0 (Basic)." W3C Recommendation. W3C. http://www.w3.org/TR/media-frags/.

Williams, Hannah. 2016. "Google's DeepMind AI Masters Lip-Reading." *Computer Business Review* (blog). November 25, 2016. http://www.cbronline.com/news/internet-of-things/googles-deepmind-ai-masters-lip-reading/.

Winograd, Terry. 2006. "Shifting Viewpoints: Artificial Intelligence and Human–Computer Interaction." *Artificial Intelligence*, Special Review Issue, 170 (18): 1256–58. https://doi.org/10.1016/j.artint.2006.10.011.

Wu, Garrett. 2013. “Why More Data and Simple Algorithms Beat Complex Analytics Models.” *Data Informed* (blog). August 7, 2013. http://data-informed.com/why-more-data-and-simple-algorithms-beat-complex-analytics-models/.

Annexes

Curriculum Vitae

Pierre-Antoine Champin
Maître de Conférences (CN) - 27ème section (Informatique)
IUT - Université Claude Bernard Lyon 1
92, bd Niels Bohr
69 622 Villeurbanne Cedex
France

Tel : +33 (0) 472 69 21 73
Fax : +33 (0) 478 93 51 56
Mel : pierre-antoine.champin at univ-lyon1 point fr

Éducation

2003 Doctorat	Université Claude Bernard Lyon 1 Mention : très honorable Dirigé par : Alain Mille, Jérôme Euzenat Titre : Modéliser l'expérience pour en assister la réutilisation : de la conception assistée par ordinateur au Web sémantique
1997 DEA	DEA d'Informatique de Lyon Mention : Assez Bien Dirigé par : Robert Laurini Titre : Recherche de chemins en milieu hostile à l'aide des projections symboliques
1997 Diplôme d'ingénieur	INSA de Lyon Spécialité : Informatique Mention : Félicitations du Jury
1992 Baccalauréat	Série C (scientifique) Mention : Bien

Expériences professionnelles

2003 – présent	Maître de conférences LIRIS UMR 5205 / Université Claude Bernard Lyon 1
2009 – 2010	Chercheur Invité Clarity / University College Dublin (République d'Irlande)
2002 – 2003	ATER Université Claude Bernard Lyon 1
2001 – 2002	Enseignant Vacataire IUT - Université Claude Bernard Lyon 1

1999 – 2002	Ingénieur doctorant CIFRE Dassault Système

Principales activités d'enseignement

Depuis mon recrutement en 2003, j'effectue mon service d'enseignement complet (192h) au département informatique de l'IUT de l'UCBL (site de Villeurbanne Doua).

J'interviens également régulièrement dans différents masters du département informatique de la FST. Enfin, j'ai participé à la création du master Architecte de l'Information (initialement co-habilité par l'ENS-Lyon et l'UCBL), où j'interviens depuis septembre 2012.

Les enseignements figurants dans le tableau ci-dessous sont ceux dont j'ai assuré la conception des supports de cours et/où la responsabilité du module.

Discipline	Années	Niveau	Établissement	Heures / an
Introduction aux technologies du Web	2011 –	L1	IUT Lyon 1	30h~
Algorithmique	2003 –	L1	IUT Lyon 1	40h~
Structures de données avancées	2003 – 2011	L1	IUT Lyon 1	40h~
Introduction à l'Intelligence Artificielle	2014 –	L2	IUT Lyon 1	13h
Programmation Web client riche	2014 –	L2	IUT Lyon 1	26h
Système d'exploitation	2003 –	L2	IUT Lyon 1	30h~
Programmation Système	2003 – 2011	L2	IUT Lyon 1	20h~
Introduction au Web de données	2012 –	M1	ENS Lyon (Master Archinfo)	12h
Programmation Web	2016 –	M2	FST Lyon 1 (Master Bio-Info)	30h
Dynamique des connaissances	2011 –	M2	FST Lyon 1 (Master IA)	30h~
Ingénierie des systèmes intelligents	2011 – 2016	M2	FST Lyon 1 (Master IADE)	12h

Pour plus d'information sur mes activités d'enseignement : http://champin.net/enseignement

Encadrement

Encadrements effectué

Étudiant	Formation	Encadrants et % d'encaftement	Dates
Rakebul Hasan	Doctorat	Fabien Gandon (80%) P-A Champin (20%)	2011 – 2014
Blandine Ginon	Doctorat	S. Jean-Daubias (50%) P-A Champin (50%)	2011 – 2014
Raphaël Cazenave-Lévêque	M2R TIW	P-A Champin (34%) Lionel Médini (33%) Amélie Cordier (33%)	2016
Raul Leaño-Martinet	M2R TIW	P-A Champin (50%) Lionel Médini (50%)	2015
Laetitia Pot	M2 ArchInfo	Christine Michel (34%) M-T Têtu (33%) P-A Champin (33%)	2014

Encadrements en cours

Étudiant	Formation	Encadrants et % d'encaftement	Dates
Fatma Derbel	Doctorat	P-A Champin (50%) Amélie Cordier (50%)	2016 –

Étudiant	Formation	Encadrants et % d'encaftement	Dates
Maxime Chabert	Doctorat	Christine Solnon (34%) P-A Champin (33%) Amélie Cordier (33%)	2015 –

Contrats de recherche

Nom du projet	Dates	Financement	Informations
Épistémè	2014 – 2017	ANR	Coordinateur LIRIS pour le projet https://projet-episteme.org/
Hubble	2014 – 2017	ANR	Responsable du lot 3 http://hubblelearn.imag.fr/
Learning Café	2013 – 2016	FUI	Coordinateur LIRIS pour le projet http://extranet.learningcafe.fr/
Spectacle en lignes(s)	2013 – 2014	ANR	Coordinateur LIRIS pour le projet http://spectacleenlignes.fr/
eGonomy	2012 – 2014	Ministère de l'industrie	http://egonomy.iri-research.org/
Kolflow	2011 – 2014	ANR	http://kolflow.univ-nantes.fr/
Cinécast	2010 – 2012	FUI	http://cinecast.fr/
ACAV	2009 – 2011	Ministère de l'industrie	http://blog.dailymotion.com/acav/
Ithaca	2008 – 2011	ANR	http://kolflow.univ-nantes.fr/
Cinélab	2007 – 2008	ANR	http://advene.org/cinelab/