Task-Centred Information Management

Tiziana Catarci[1], Alan Dix[2], Akrivi Katifori[3], Giorgios Lepouras[4], and Antonella Poggi[1]

[1] Dipartimento di Informatica e Sistemistica,
Sapienza Università di Roma,
Via Salaria 113, 00198 Roma, Italy
{poggi,catarci}@dis.uniroma1.it
[2] Computing Department,
Lancaster University,
Lancaster, LA1 4YR, United Kingdom
alan@hcibook.com
[3] Dept. of Informatics and Telecommunications,
University of Athens
Panepistimiopolis, 157 84, Athens, Greece
vivi@mm.di.uoa.gr
[4] Dept. of Computer Science,
University of Peloponnese,
Tripolis, Greece
gl@uop.gr

Abstract. The goal of DELOS Task 4.8 Task-centered Information Management is to provide the user with a Task-centered Information Management system (TIM), which automates user's most frequent activities, by exploiting the collection of personal documents. In previous work we have explored the issue of managing personal data by enriching them with semantics according to a Personal Ontology, i.e. a user-tailored description of her domain of interest. Moreover, we have proposed a task specification language and a top-down approach to task inference, where the user specifies main aspects of the tasks using forms of declarative scripting. Recently, we have addressed new challenging issues related to TIM user's task inference. More precisely, the first main contribution of this paper is the investigation of task inference theoretical issues. In particular, we show how the use of the Personal Ontology helps for computing simple task inference. The second contribution is an architecture for the system that implements simple task inference. In the current phase we are implementing a prototype for TIM whose architecture is the one presented in this paper.

1 Introduction

Personal Information Management (PIM) aims at supporting users in the collection, storage and retrieval of their personal information. It is a crucial challenge nowadays; indeed, the collection of digital documents stored within the personal repository of each one of us increases every day, in terms of both size and "personal" relevance. In a sense, this collection constitutes the Digital Library that is closest to the user and most

C. Thanos, F. Borri, and L. Candela (Eds.): Digital Libraries: R&D, LNCS 4877, pp. 197–206, 2007.

commonly and frequently used. It is often the starting point or reason for wider access to digital resources. Furthermore, the user's personal document collection is daily used to perform routine activities, as booking hotels, and organising meetings. The focus of DELOS Task 4.8 Task-centred Information Management is precisely to provide the user with a Task-centred Information Management system (TIM), which automates these user activities, by exploiting the collection of personal documents considered as a Digital Library.

At the beginning of our investigation, we have faced the issue of providing the user with a system allowing her to access her data through a Personal Ontology (PO), that is unified, integrated and virtual, and reflects the user's view of her own domain of interest [3,10]. The study of such a system is motivated by the need to assign semantics to the user's personal collection of data, in order to help inferring and executing the most frequent user's tasks that either take such data as input or produce it as output. Then, we have focused on defining a task specification language [4], which was appropriate in our context, in the sense that it was both expressive enough to be effectively helpful to the user, and simple enough to allow tasks to be inferred. Moreover, such a language had to be tailored to use the PO to semi-automatically obtain task input and to possibly update the PO after the task execution, according to the task output and semantics. However, having a task specification language allows to semi-automatically execute tasks that the user previously defined. Our further goal is to have the system infer what is the task the user intends to execute next.

This paper addresses the above mentioned issues. Specifically, our main contributions can be summarised as follows:

- First, we cope with task inference in TIM. In particular, we investigate the main task inference theoretical issues. Then we show how to exploit the underlying PO and the features of our task specification language, in order to provide simple task inference in our setting. The main idea here is to suggest appropriate tasks from personal data, based on their semantics.
- Second, we propose a novel architecture for TIM, which implements our solution to the simple task inference issue. More precisely, we propose to integrate into the system an existing web bookmarking tool, called *Snip!t* , whose distinguishing feature is precisely to "intelligently" suggest appropriate actions to perform starting from a Web page content.

The paper is organised as follows. After discussing related work in Section 2, we briefly introduce in Section 3 the previous main task contributions. In Section4 we discuss task inference related theoretical issues. Then, in Section 5, after introducing *Snip!t* and presenting its distinguishing features, we propose a new architecture for TIM, and show how this provides a simple task inference. Finally, we conclude by presenting future work.

2 Related Work

The issues faced in DELOS Task 4.8 concern several different areas of research. In the following we discuss related work in each of the main areas.

Task management. Recently, research efforts on the problem of managing user's tasks have lead to the prototype Activity-Centered Task Assistant (ACTA), implemented as a Microsoft Outlook add-in [1]. In ACTA, a user's task, named "ACTA activity", is represented as a pre-structured container, that can be created inside the e-mail folder hierarchy. It is a task-specific collection containing structured predefined elements called "components", that embody common resources of the task and appear as activity subfolders. Thus, for example, by creating an ACTA activity to represent a meeting, and by inserting the component "contacts", the user aims at relating the content of this subfolder, which will essentially be a list of names, with that particular meeting. Moreover, the population of an activity is done semi-automatically, by allowing the user just to drag into the appropriate activity component, the document containing the relevant data, which is afterward automatically extracted and stored. Even though ACTA activities are built relying on user's personal data, their approach is not comparable to ours, since they do not consider tasks as a workflow of actions (e.g. filling and sending the meeting invitation e-mail), which can be inferred and semi-automatically executed.

Task inference. There has been a long history of research into task detection, inference and prediction in human-computer interaction, with a substantial activity in the early 1990s including Alan Cypher's work on Eager [5] and several collections [8]. The line of work has continued (for example [11]), but with less intensity than the early 1990s. Forms of task inference can be found in widely used systems, for example the detection of lists etc. in Microsoft Office or web browsers that auto-fill forms. The first example clearly demonstrates how important it is that the interaction is embedded within an appropriate framework, and how annoying it can be when inference does not do what you want! Some of this work lies under the area of "programming by demonstration" or "programming by example", where the user is often expected to be aware of the inferences being made and actively modify their actions to aid the system. This is the case of [9] where authors present a learning system, called PLIANT, that helps users anticipating their goal, by learning their preferences and adaptively assisting them in a particular long-running application such as a calendar assistant. Other work falls more under user modelling, intelligent help, automatic adaptation or context-aware interfaces where the user may not be explicitly aware that the system is doing any form of inference [2]. Our work lies with the former as we do expect that users will be aware of the inferences being made and work symbiotically with the system in order to create a fluid working environment.

3 Summary of Previous Contributions

In this section we briefly present the DELOS Task 4.8 previous contributions, namely OntoPIM and the task specification language.

OntoPIM. OntoPIM [10] is a module that allows to manage the whole collection of heterogeneous personal data usually maintained in a personal computer (e.g. contacts, documents, e-mails), and to access them through a unified, integrated, virtual and yet user-tailored view of her data. This view is called Personal Ontology (PO), since it reflects the user's view of her own domain of interest. It is therefore specific to each user.

As for the language to specify the PO, we use the Description Logic called *DL-Lite*$_A$ [13,12], since besides allowing to express the most commonly used modelling constructs, it allows answering expressive queries, i.e. conjunctive queries, in polynomial time with respect to the size of the data. This is clearly a distinguishing and desirable feature of such a language, in a context like ours, since the amount of data is typically huge in one's personal computer. In order to achieve the above mentioned result, OntoPIM proceeds as follows. First it extracts, by means of appropriate wrappers, pieces of relevant data from the actual personal data contained in the personal computer. Then it exploits the so-called Semantic Save module, which *(i)* stores such data in a DBMS, maintaining also its provenance, and *(ii)* stores the relationship existing between the data and the PO, as (implicitly) specified by the user. Note that the latter relationship reflects indeed the data semantics according to the user.

Task Specification. In order to be able to semi-automatically execute user tasks, we defined a task specification language [4] having two main features. First, the language is at the same time expressive enough for actually being helpful to the user, and simple enough for being effectively "usable" and "learnable" by the system. Second, the language allows to specify as a part of the task definition, the input/output data mappings, i.e. the relationships existing between the task and the PO. Specifically, the input data mappings specify the query to be posed over the PO in order to obtain the task input, whereas the output data mapping specify the task output as an update (possibly empty) to be computed over the personal data, according to the semantics of both the task execution and the PO. As we will see, the specification of task input/output data mappings is crucial for task inference/detection/suggestion.

Furthermore, we have explored task inference top-down approaches, where the user specifies aspects of the task using forms of declarative scripting. Our proposal was based on the idea of combining task decomposition and a plan language to describe for each complex task, the execution plan of its sub-tasks. On one hand, a complex task is decomposed into a set of sub-tasks. This allows for a comprehensible view of the task. On the other hand, we have proposed a plan language limited to sequence, alternatives and repetition.

4 Task Inference

In this section, we first address task inference theoretical issues. In particular we focus on bottom-up approaches to simple task inference. Then, we show how the use of a PO can help solving simple task inference in our setting.

As discussed in Section 2, the most successful systems have often been within dedicated applications, where there is a single line of activity and detailed semantic understanding of each action. This was for example the case with EAGER and PLIANT. For more generic systems detection and inference is more difficult, particularly where the user trace data is at a low level either keystroke or click-stream data (which often has to be the case where application support is not assumed). In fact, two particular problems for generic systems are:

– *interleaving*. Users are often performing several tasks simultaneously, perhaps while waiting for something to complete, or because they notice and alert, get a

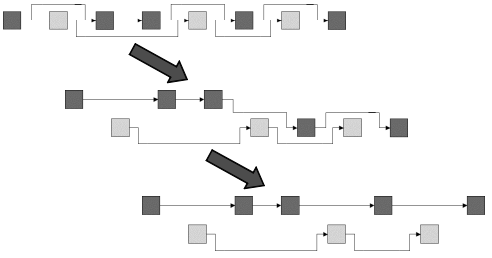

Fig. 1. Untangling interleaved tasks using dependencies

telephone call. Before task inference can begin it is necessary to disentangle these, otherwise each sub-task is littered with the "noise" of the others.

– *generalisation.* Where the user entered data is simply sequences of keystrokes, clicks on locations or basic data types, it is hard to generalise without very large numbers of examples. The use of the PO helps considerably with these problems. Specifically, to face the interleaving problem, we will encourage a drill-down model of interaction where the user either selects previous outputs of actions and then drills forwards (e.g. recent e-mail → sender → Antonella → University → City → Rome → Flights to Rome) or responds to suggested form fills (e.g. from Flight booking form with field 'City' select "Rome because it is the City where Antonella works"). This creates an explicit link either directly between actions or indirectly between them through ontology relationships, which can then be used to separate out interleaved tasks and sub-tasks by tracing lines of dependency, rather like pulling out a string of pearls from a jewellery box (see Figure 1).

Concerning the generalisation problem, because we have a rich typing of task/action input and output data through the PO, we are in a much better position to generalise. If we only know that a form field requires a character string, then given a future character string we may have many possible previous tasks sequences whose initial actions require a string. In contrast, if we know that a character string is in fact a person name or a city name, then faced with a future person name (perhaps from a directory look-up, or an e-mail sender) it is easier to find past tasks requiring as input a person name. In other words, our generalisation is not based on the representation in terms of letters, but in terms of the elements of the ontology.

Let us now focus on simple task inference, where a simple task can include sequences and/or a repetitions of sub-tasks/actions. Thus, here we do not cope with choices. Hence, the job of the inference here is to suggest the most likely single actions and entire task sequences so as to minimise the user's effort in using the system. Intuitively, we intend to build a series of increasingly complex inference mechanisms, both in terms of our development process and in terms of the users' experience. That is, even if we have complex inference mechanisms available, these need to be presented incrementally to the user. In fact, to some extent, even simple type matching can be viewed as a crude form of task inference. However, if this is supplemented by sorting of a few most likely candidate actions/tasks based on past usage, then a form of task sequence comes almost "for free".

For example, suppose that the user has recently (1) taken a phone number, (2) done a reverse directory look-up (in some external web service) to find the person's name, then (3) done an address look-up to find their address and finally (4) taken the address and used a web mapping service to show the location. Now, suppose the user has an e-mail containing a telephone number. The content recogniser finds the number and so suggests a semantic save of the number and also actions using the number. Top of the action list would be likely actions on telephone numbers, and notably the reverse look-up because it was recent. Once this was done, one of the outputs of the reverse look-up would be the person's name and similarly its top option would be the address look-up. So at each stage the previous sequence would be the first option, which means that the task is not totally automated, but little user effort is required. The next step, which likewise requires minimal "intelligence", is simply to offer the entire sequence of actions as one of the options when the telephone number is encountered. This simply requires that all recent/previous task sequences are stored and so recent or frequently performed task sequences starting with the given type are suggested. The user is also given the option of naming a particular sequence of actions. If the user chooses to do this, the task can be used in future interactions. Note too that selection and more so naming is effectively a confirmation by the user that this is a useful task sequence and so will make it far more likely to be presented in future.

More complex tasks including repetitions of sub tasks can similarly be built bottom-up, for example, if the user chooses to perform an action sequence on an instance that is in some visible collection (e.g. selected from a PO concept, or from table of results) and then proceeds to select a second instance in the collection, one of the options would be not only to perform the sequence on the selected item but on all the collection (or selected members of it).

5 *Snip!t* and TIM

In this section, we first present an existing tool, called *Snip!t* , that is a web bookmarking tool, whose distinguishing feature is to suggest appropriate actions to perform starting from a Web page content. Then we propose a novel architecture for TIM, implementing task inference as discussed in Section 4, by integrating *Snip!t* into the system.

Snip!t [6] is a web bookmarking tool, however unlike most web bookmarks, the user can also select a portion (snip) of the page content and then, using a bookmarklet, this selection is sent to the *Snip!t* web application. The snip of the page is stored along with the page url, title etc. and the user can sort the snip into categories or share it with others using RSS feeds. In addition, when the selected text contains a recognised type of data such as a date, post code, person's name, etc., then actions are suggested. For example, if the selected text is recognised as a post code and this leads to suggested actions such as finding the local BBC news for a specific area (see Figure 2).

Snip!t had two contributing origins. In a study of bookmarking organisation some years ago [7] some subjects said that they would sometimes like to bookmark a portion of a page. While the study had different aims this suggestion led to the first version of *Snip!t* in 2003. In addition, this gave an opportunity to use the new platform for data-detector technology originally developed as part of *onCue*, the key product of an

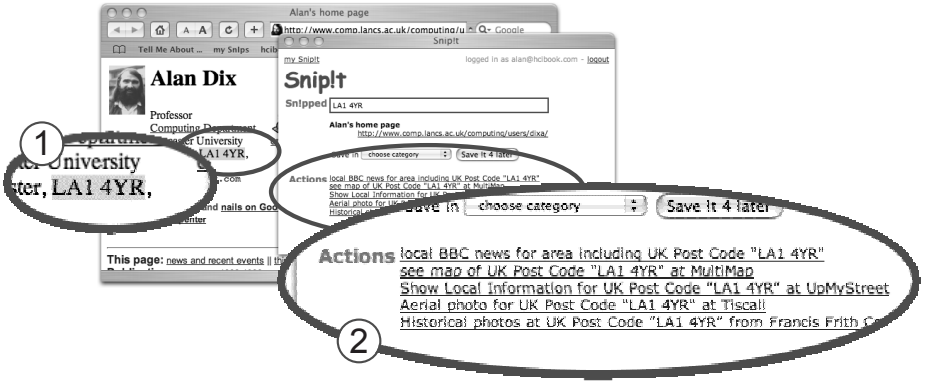

Fig. 2. *Snip!t* in action (actions for a post code)

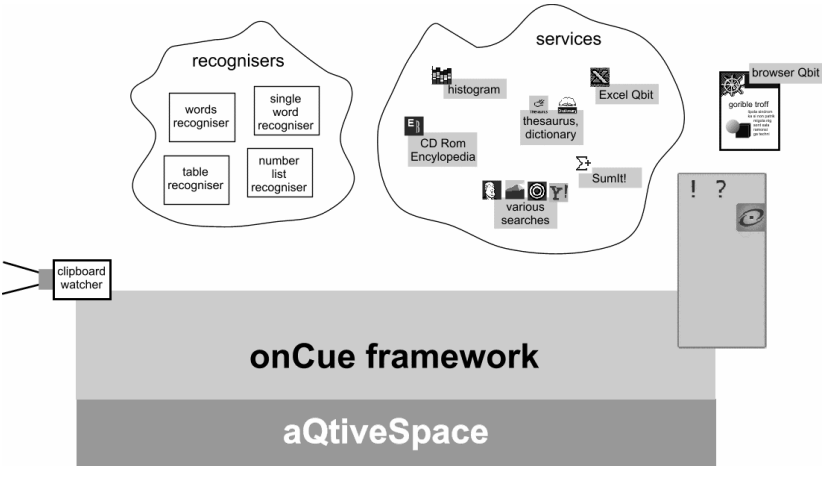

Fig. 3. *onCue* architecture

ex-dot.com company aQtive [6]. So *Snip!t* combines a way of storing and sorting data (portions of web pages) and intelligent suggestion of actions to perform based on the data. Thus it shares some characteristics with the TIM vision.

Internally the bookmarking side of *Snip!t* is fairly straightforward with simple hierarchical categorisation scheme and the ability to assign snips to multiple categories. The snipping of the page contents itself is done using a bookmarklet, that is a small piece of Javascript that is placed as a browser bookmark, usually on the browser toolbar. When the user clicks the bookmarklet the Javascript executes extracts the selected content and then creates a HTTP request to the *Snip!t* server.

The ”intelligent” parts of *Snip!t* use an architecture inherited from *onCue*. It consists of two main types of agents: recognisers and services (cf. Figure 3). Recognisers work

on plain text and discover various data types whilst services take these data types and suggest actions based on them (usually creating invocations of web applications). The recognisers within *Snip!t* are of several types:

- basic recognisers: can be based either on large table look-up, e.g. common surnames and forenames, place names, or on regular expression / pattern matching, e.g. telephone number, ISBN.
- chained recognisers: where the semantics of data recognised by a recogniser is used by another to:
  - look for a wider data representation, e.g. postcode suggests looking for address; these recognisers are used to allow scalability and reduce false positives;
  - look for a semantically related data, e.g. URL suggests looking for the domain name; these recognisers are used to reduce false negatives;
  - look for inner representation, e.g. from Amazon author URL to author name; these recognisers are also used to allow scalability.

Each agent, recogniser and service, is relatively simple, however the combinations of these small components create an emergent effect that intelligently suggests appropriate actions based on the snipped text. Let us now turn attention to TIM, and the way the user interacts with the system in order to execute a task. The starting point may be of three kinds:

(i) explicit invocation of a particular task (e.g. selecting it from an application menu),
(ii) choosing a data value in a document, e-mail etc. then drilling down into tasks possible from it,
(iii) selecting an item in the PO and then, drilling into available tasks (that is one's with matching types).

Consider now the architecture illustrated in Figure 4. The main idea is to integrate *Snip!t* within the system. In particular, in case *(i)* the user will directly indicate among all *Snip!t* services the one he wants to execute next. In contrast, cases *(ii)* and *(iii)* are closer to the typical *Snip!t* scenario. Indeed, in both cases, given a (collection of) data value(s), the system suggests tasks to perform from it. Apart from OntoPIM and *Snip!t*, this new architecture includes the three main TIM modules discussed below.

- **TIM application wrapper:** This module, which is dependent on the particular application enables the user to select any piece of document and send it to *Snip!t* in an appropriate form (cf. snip S). As for a demonstration, we have extended the functionality of Thunderbird mail client, so as to allow the user to select the TIM button while reading a message. A script running on the mail client saves the message, parses the MIME headers filling-in an HTML form which appears in a new browser window. The user can then press the submit button to send the message to *Snip!t*.
- **Personal Ontology Interaction Module (POIM):** Given a collection of data W each with its associated type as returned by *Snip!t* recognisers, this module access the PO and returns a collection of data W' somehow related to W according to the PO. Such data is then used to perform appropriate tasks.
- **Task Inferencer:** This module is responsible for task inference. Intuitively, given a collection of data W' from the POIM and its provenance, the Task Inferencer suggests what is the next task/action to be executed from W.

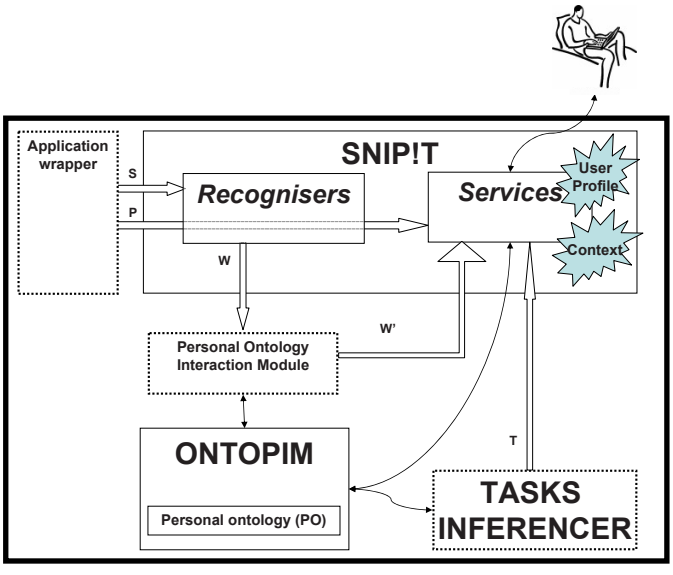

Fig. 4. Integrating *Snip!t* into a personal task-centric system

6 Conclusion and Future Work

In this paper we have investigated new challenges toward a fully equipped TIM system. In particular, we have addressed theoretical task inference issues, and we have proposed a first implementation providing a solution to simple task inference in one's personal computer provided with a PO.

Many other aspects deserve to be further investigated, in particular concerning task inference. Indeed, we have concentrated on simple tasks, possibly including sequences and repetitions of actions, whereas we have not coped with alternatives, i.e. choices. These may arise either where a selection has to be made from a collection (e.g. $1 - m$ relations in the ontology) or where tasks sequences vary slightly depending on aspects of the context or instance (e.g. compress files larger than 100K). Various machine learning techniques can be applied here, which we plan to study in future work.

References

1. Bellotti, V., Thornton, J.: Managing activities with tv-acta: Taskvista and activity-centered task assistant. In: PIM 2006. Proc. of the Second SIGIR Workshop on Personal Information Management (2006)
2. Boone, G.: Concept features in re: Agent, an intelligent email agent. In: Proc. of the Second international Conference on Autonomous Agents, pp. 141–148. ACM Press, New York (1998)
3. Catarci, T., Dong, X., Halevy, A., Poggi, A.: Structure everything. In: Jones, W., Teevan, J. (eds.) Personal Information Management, University of Washington Press (UW Press) (to appear)

4. Catarci, T., Habegger, B., Poggi, A., Dix, A., Ioannidis, Y., Katifori, A., Lepouras, G.: Intelligent user task oriented systems. In: PIM 2006. Proc. of the Second SIGIR Workshop on Personal Information Management (2006)
5. Cypher, A.: Eager: Programming repetitive tasks by example. In: CHI 1991. Proc. of the Conference on Human Factors in Computing Systems, pp. 33–39 (1991)
6. Dix, A., Beale, R., Wood, A.: Architectures to make Simple Visualisations using Simple Systems. In: AVI 2000. Proc. of Advanced Visual Interfaces, pp. 51–60 (2000)
7. Dix, A., Marshall, J.: At the right time: when to sort web history and bookmarks. In: Proc. of HCI International 2003, vol. 1, pp. 758–762 (2003)
8. Finlay, J., Beale, R.: Neural networks and pattern recognition in human-computer interaction. ACM SIGCHI Bulletin 25(2), 25–35 (1993)
9. Gervasio, M.T., Moffitt, M.D., Pollack, M.E., Taylor, J.M., Uribe, T.E.: Active preference learning for personalized calendar scheduling assistance. In: IUI 2005. Proc. of the 10th international conference on Intelligent user interfaces, pp. 90–97. ACM Press, New York (2005)
10. Katifori, V., Poggi, A., Scannapieco, M., Catarci, T., Ioannidis, Y.: OntoPIM: how to rely on a personal ontology for Personal Information Management. In: Proc. of the 1st Workshop on The Semantic Desktop (2005)
11. Lieberman, H.: Your wish is my command: programming by example. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
12. Poggi, A.: Structured and Semi-Structured Data Integration. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza" (2006)
13. Poggi, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Linking data to ontologies. Submitted for publication to an international journal (2007)