Semantic Templates for Designing Recommender Systems*

Juan A. Recio-García[1], Belén Díaz-Agudo[1]
Derek Bridge[2], Pedro A. González-Calero[1]

[1] Dept. Software Eng. and Artificial Intelligence
Universidad Complutense de Madrid, Spain
email:jareciog@fdi.ucm.es,
{belend,pedro}@sip.ucm.es

[2] Dept. of Computer Science
University College Cork, Ireland
d.bridge@cs.ucc.ie

Abstract

In this paper we describe ongoing research into a flexible way of designing CBR systems in jCOLIBRI 2 using a library of *templates* obtained from a set of previously designed CBR systems (i.e. a case base of CBR design experience). In case-based fashion, jCOLIBRI will retrieve templates from the library; the designer will choose one template, and adapt it. jCOLIBRI will also suggest suitable substitutions from the semantic annotations in the components. In this paper we focus on one of the main bottlenecks of this approach: obtaining and representing an appropriate set of templates. We are working on templates within a successful set of CBR systems: case-based recommender systems and other memory-based recommender systems. We also describe the graphical template editor and how we represent the templates using Semantic Web technologies.

Keywords: CBR, jCOLIBRI, design of CBR systems, recommender systems

1. Introduction

Developing a CBR application from scratch is a hard task. However, the design of a CBR application could be easier if some of the components of the new application could be reused from previous developments. Based on this idea, the Madrid research group has developed jCOLIBRI, an object-oriented framework in Java for building CBR systems that greatly benefits from the reuse of previously developed CBR systems.

In (Recio et al. 2006) we have reviewed the advantages and drawbacks of the jCOLIBRI framework (version 1) from the experience of two years of development. The recently released new version of the framework (jCOLIBRI 2) tries to solve many of the problems identified in the previous version. jCOLIBRI version 2 is a new implementation that follows a new and clear architecture divided into two layers: one oriented to developers (released in September 2007), the other oriented to designers (still in progress).

* Supported by the Spanish Committee of Science & Technology (TIN2006-15140-C03-02)

For the top layer aimed at designer users we are working on the semi-automatic composition of systems from components. We propose a flexible way to design CBR systems in jCOLIBRI 2 using a library of *templates* obtained from a previously designed set of CBR systems. In case-based fashion, jCOLIBRI will retrieve templates from a library of templates (i.e. a case base of CBR design experience); the designer will choose one, and adapt it. jCOLIBRI will also suggest suitable substitutions from the semantic annotations in the components. We are aiming for a semi-automatic way of designing CBR systems where the designer interacts with jCOLIBRI to configure the system. For the present at least, we consider *fully* automatic design of CBR systems to be unachievable. It would require a very rich semantic mark-up of the reusable components. Moreover it would require a very expressive language to let the designer provide the system with a detailed description of the system (s)he wants to design.

Using jCOLIBRI (version 2 top layer) and a library of templates, to design a new CBR system will be a process similar to the CBR cycle itself:

1. The designer writes a query to explain the CBR system (s)he wants to build.
2. jCOLIBRI retrieves one or more templates that are suitable for this query. Similarity is based on the semantic annotation of the templates and their components. For now, we assume that one of the retrieved templates is selected by the user.
3. Using the semantic annotations of the components of the selected template, jCOLIBRI guides the user in the manual adaptation of the template. The designer is in charge of configuring the data sources and any domain-specific similarity measures that are not already available in jCOLIBRI.
4. The resulting system is tested; it may be revised and then it may be abstracted into a new template that is retained in the template library.

Our approach has one important bottleneck: the acquisition of an initial design experience template case base. To begin with, we have limited our system to work with a well known and successful family of systems: recommender systems. A recommender system infers the goals and preferences of its user; it uses the inferred knowledge to select and/or rank products, services or information sources (generically called *items*); and it recommends to its user items that may satisfy the inferred goals and meet the inferred preferences.

This rest of this paper is structured as follows. Section 2 explains the idea of template-based design and its main difficulties and open lines of work. We focus on the issues of template acquisition and representation. Section 3 presents our first attempt at a case base of templates for recommender systems. Section 4 describes the template editor and how it represents the templates using Semantic Web technologies. We review our main conclusions and lines of future work in Section 5.

2. Template-Based Design in jCOLIBRI 2

In jCOLIBRI 1, design was based on decomposing a system's reasoning into tasks and methods for achieving those tasks. Where a task was decomposed into more than one subtask, jCOLIBRI 1 imposed the restriction that the subtasks be combined in a linear sequence (no conditionals, no iteration). Template-based design in jCOLIBRI 2 is based on the same underlying idea of decomposing the reasoning into tasks and methods. However, there is no longer the restriction that subtasks be composed only as a linear sequence; in templates, more complicated flow of control is allowed. Each template can be understood as a generalization of several CBR systems and needs to be defined carefully in collaboration with CBR experts.

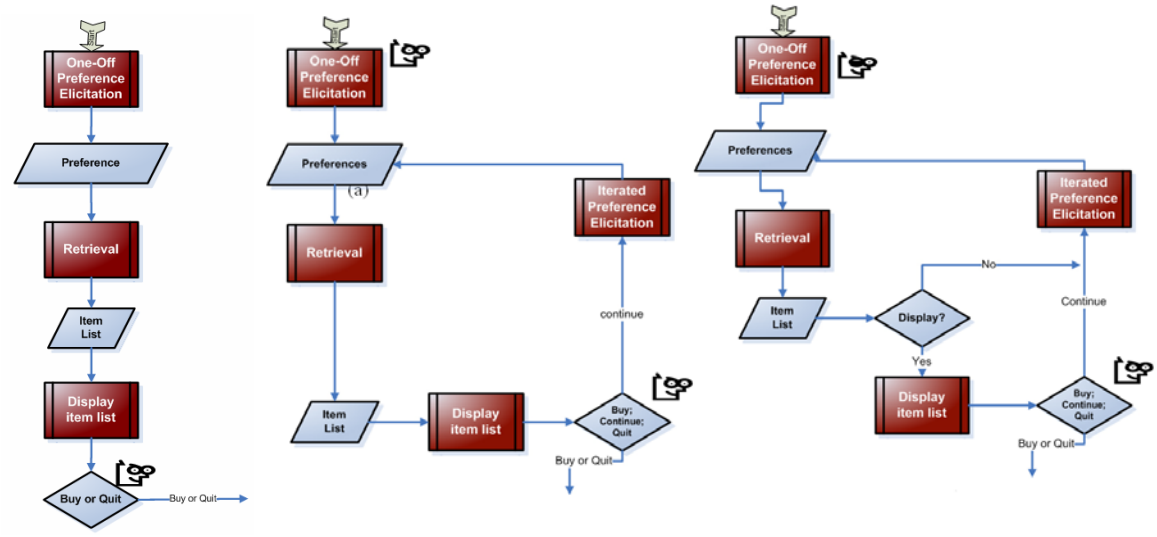

Figure 1. Single Shot Systems (left), Conversational A (middle) and B (right)

We represent templates graphically as shown in Figure 1. Each rectangle in the template is a subtask. Simple tasks (shown as blue or pale grey rectangles) can be solved directly by a resolution method. Complex tasks (shown as red or dark grey rectangles) are solved by decomposition methods having other associated templates. There may be multiple alternative methods to solve any given task.

This approach poses a number of interesting research challenges:

1. How to obtain a set of templates that are representative enough. Having a case base of templates requires thinking about granularity, level of generality, diversity, and coverage of the template case base. Besides we need to represent dependencies between the elements of a template, and annotate the templates and their components. Each template has different variability points, i.e., tasks that can be solved in different ways using different methods.
2. How to define the query vocabulary to retrieve templates. How to explain the design requirements is an open question. We envisage a process where the system asks questions to the system designer to guide the query definition, and where queries will be biased towards typical processes and types of user interaction, not towards lower-level details about, for example, case representation. This is because in template adaptation it is easier to change these low-level details than it is to modify the overall workflow of the template.
3. How to define a similarity metric between the query and templates based on the semantic annotations of the components.
4. How to adapt templates. The question that arises here is how easy it will be to (manually) adapt a template to fulfil the requirements in the query. Although it is a difficult process the system can help by taking into account dependencies, and suggesting substitutions.

In order to support these four processes, we are working on the semantic annotation of components and templates using vocabulary from CBROnto (Díaz and González 2002). The CBROnto ontology formalizes the syntactic, semantic and pragmatic aspects of the reusable components of the framework.

Using the CBROnto vocabulary we can annotate the components and reason about their composition. This will facilitate the automatic configuration of complex systems.

In this paper we focus on the first of these challenges: obtaining and representing a set of templates. As the CBR community has produced many different families of systems, we begin by narrowing the scope of our template library to cover one of the most successful families: recommender systems, especially case-based recommender systems. Our first attempt at a set of templates is presented in Section 3. Section 4 describes the template editor used to draw, and represent these templates in a formal language that will support template selection and adaptation.

3. Obtaining Templates for Recommender Systems

As a case study, we have done an analysis of recommender systems that is based in part on the conceptual framework described in the review paper by Bridge *et al*. (2006). The framework distinguishes between collaborative and case-based, reactive and proactive, single-shot and conversational, and asking and proposing. Within this framework, the authors review a selection of papers from the case-based recommender systems literature, covering the development of these systems over the last ten years.

We take the systems' interaction behaviour as the fundamental distinction from which we construct templates:

- *Single-Shot Systems* make a suggestion and finish. Figure 1 (left) shows the template for this kind of system, where *One-Off Preference Elicitation* and *Retrieval* are complex tasks that are solved by decomposition methods having other associated templates.

- After retrieving items, *Conversational Systems* (Figure 1 middle and right) may invite or allow the user to refine his/her current preferences, typically based on the recommended items. *Iterated Preference Elicitation* might be done by allowing the user to select and critique a recommended item thereby producing a modified query, which requires that one or more retrieved items be displayed (Figure 1 middle). Alternatively, it might be done by asking the user a further question or questions thereby refining the query, in which case the retrieved items might be displayed every time (Figure 1 middle) or might be displayed only when some criterion is satisfied (e.g. when the size of the set is 'small enough') (Figure 1 right). Note that both templates share the *One-Off Preference Elicitation* and *Retrieval* tasks with single-shot systems.

The variability of the templates is specified as the existence of different methods to solve a task of the template. Complex tasks (red or dark grey rectangles) are solved by decomposition methods. In this case, other templates may be defined to solve these tasks. We now present the templates we have defined for the three complex tasks to be found in Figure 1. Ultimately, our goal is to additionally annotate each method with semantic information. These annotations will help inexperienced system designers to choose the right methods for their domain. Complex tasks appearing in the templates of Figure 1 are *One-Off Preference Elicitation*, *Retrieval* and *Iterated Preference Elicitation*.

3.1 One-Off Preference Elicitation

We can identify three templates by which the user's initial preferences may be elicited:

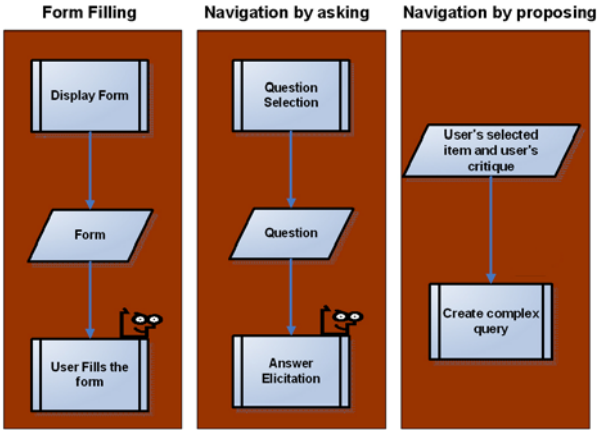

Figure 2: Methods for solving the One-Off and Iterated Query Elicitation tasks.
(a) Form-Filling, (b) Navigation-by-Asking, and (c) Navigation-by-Proposing

- One possibility is *Profile Identification* where the user identifies him/herself, e.g. by logging in, enabling retrieval of a user profile from a profile database.
- An alternative is *Initial Query Elicitation*. This is itself a complex task with multiple alternative decompositions. The decompositions include: *Form-Filling* (Figure 2a) and *Navigation-by-Asking* (i.e. choosing and asking a question) (Figure 2b). Various versions of the Entrée system (Burke 2002) offered interesting further methods: *Identify an Item* (where the user gives enough information to identify an item that s/he likes, e.g. a restaurant in his/her home town, whose description forms the basis of a query, e.g. for a restaurant in the town being visited); and *Select an Exemplar* (where a small set of contrasting items is selected and displayed, the user chooses the one most similar to what s/he is seeking, and its description forms the basis of a query).
- The third possibility is *Profile Identification & Query Elicitation*, in which the previous two tasks are combined.

For lack of space, we cannot give diagrams for most of the templates mentioned in this section.

3.2 Retrieval

Because we are focussing on case-based recommender systems (and related memory-based recommenders including collaborative filters), *Retrieval* is common to all our recommender systems. *Retrieval* is a complex task, with many alternative decompositions.

The following is a non-exhaustive list of papers that define methods that can achieve the *Retrieval* task: Wilke *et al.* 1998 (similarity-based retrieval using a query of preferred values); Smyth & McClave 2001 (diversity-enhanced similarity-based retrieval); McSherry 2002 (diversity-conscious retrieval); Bridge & Fergsuon 2002 (order-based retrieval); McSherry 2003 (compromise-driven retrieval); Bradley & Smyth 2003 (where user profiles are mined and used); Herlocker *et al.* 1991 (user-based collaborative filtering); Sarwar *et al.* 2001 (item-based collaborative filtering). In all these ways of achieving *Retrieval*, a scoring process is followed by a selection process. For example, in similarity-based retrieval (*k*-NN), items are scored by their similarity to the user's preferences and then the *k* highest-scoring items are selected for

display; in diversity-enhanced similarity-based retrieval, items are scored in the same way and then a diverse set is selected from the highest-scoring items; and so on. For lack of space, we cannot show a diagram that decomposes the *Retrieval* task in this way. Note also that there are alternative decompositions of the *Retrieval* task that would not have this two-step character..

3.3 Iterated Preference Elicitation

In *Iterated Preference Elicitation* the user, who may or may not have just been shown some products (Figure 1), may, either voluntarily or at the system's behest, provide further information about his/her preferences. Alternative decompositions of this task include:

- *Form-Filling* where the user enters values into a form that usually has the same structure as items in the database (Figure 2a). We have seen that *Form-Filling* is also used for *One-Off Preference Elicitation*. When it is used in *Iterated Preference Elicitation*, it is most likely that the user edits values s/he previously entered into the form.

- *Navigation-by-Asking* is another method that can be used for both *One-Off Preference Elicitation* and for *Iterated Preference Elicitation*. The system refines the query with the user's answer to a question about his/her preferences. The system uses a heuristic to choose the next best question to ask. Bergmann reviews some of the methods that have been used to choose this question (Bergmann 2002).

- *Navigation-by-Proposing* (also known as tweaking and as critiquing) requires that the user has been shown a set of candidate items. S/he selects the one that comes closest to satisfying his/her requirements but then offers a critique (e.g. "like this but cheaper"). A complex query is constructed that is intended to retrieve items that are similar to the selected item but which also satisfy the critique. The selection of the candidate item and its critiques must be performed during the *Display Item List* task. Therefore, the *Create Complex Query* task will receive that information and modify the query according to the user selection. To maintain the coherence of the template, our template editor must check that the method solving *Display Item List* returns both the selected item and the critiques.

4. Template Editor in jCOLIBRI

The template-based design process described in Section 2 requires a case base of templates of CBR systems, represented in a formal language that allows us to measure similarity between the query and the templates, and to adapt the template to the query requirements. In Section 3 we have informally described a case base of templates for recommender systems. However, in order to work with these templates we need to incorporate them into jCOLIBRI. We have developed a graphical template editor to create and save the templates of CBR systems.

There are many formalisms for representing workflow templates. We use *Semantic Web Services (SWS)*. In the SWS community there are different standards to represent the behaviour of software components and their composition. Examples are OWL-S[1] and WSMX[2]. Both formalisms have in common the use of

[1] Semantic markup of web services: http://www.w3.org/Submission/OWL-S/

[2] Web service execution environment: http://www.wsmx.org/people.html

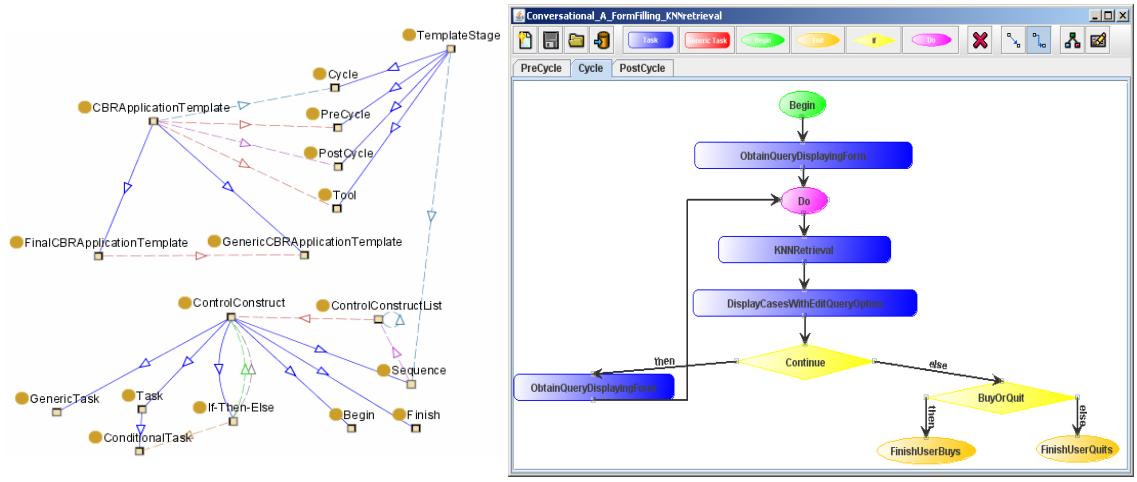

Figure 3. CBROnto (left) and Template Editor (right).

ontologies to represent the information about components and their composition, a feature that fits perfectly in our system because we already use an ontology for CBR systems: CBROnto (Díaz and González, 2002). As it stood, CBROnto provided the vocabulary to describe the methods (components) of the template framework, but it lacked a way to describe more complex control flow. We have now added the vocabulary needed to represent this aspect of the templates. For this, we use the approach of the OWL-S ontology, where several concepts are specially designed to represent workflow. Our choice was pragmatic: CBROnto is already represented using the OWL ontology language, and OWL-S also uses this language.

The vocabulary that we have added is shown in Figure 3 left. In this figure, solid lines denote subclassing and dotted lines represent properties. A template (*CBRApplicationTemplate*) can be *Generic* or *Final*. Generic templates have at least one generic task and final templates are only composed of simple tasks. Each template is divided into three main stages: the Precycle (code to initialize the application), Cycle (main CBR cycle) and Postcycle (post execution code). Moreover, a CBR application can have tools used for maintenance and other similar purposes. Each application stage contains a sequence of elements: a Begin Task, a Final Task, Generic Tasks, Conditions and other subsequences. A sequence is composed of a Control Construct List that contains Control Constructs. Conditionals (If-Then-Else) are defined with a Conditional Task (a task that returns a Boolean value) and two Control Construct branches that correspond with the *then* and *else* branches. All these concepts are direct adaptations of the OWL-S ontology. Our template graphical editor (Figure 3 right) allows us to create templates graphically and save them using the OWL formalism.

Before creating a new template, the user must introduce some metadata: template name, authors, type, generic or final template, additional tools, and so on. Then, each workflow can be created in a graphical way. When the user creates a new Task or Generic Task element, the tool reads the instances contained in CBROnto that belong to these concepts. Those instances represent the tasks solved by the methods of the framework and are included as new components.

5. Conclusions and Future Work

In this paper we have described a new approach to composing CBR systems based on templates. Templates are obtained from experts and define stereotypical CBR applications. We formalize them in the OWL language using vocabulary from an extended version of CBROnto: an ontology for CBR components and applications that takes advantage of previous work in Semantic Web Services.

We are working on the development of the jCOLIBRI semi-automatic composition tools. We are currently finishing the development of the methods used to solve the tasks in the recommender system templates. Once the library has been populated with all these methods, we will work on the tool to guide the designers in the creation of CBR systems. This tool can be seen as a CBR application with a case base composed of templates that are adapted depending on the requirements of the users. In this way, the system recommends a template and the user completes that template by assigning a method for each task. To perform this task-filling process, the tool will use annotations about inputs, outputs, preconditions and postconditions of CBROnto that describe each component of the framework.

References

Bergmann, R (2002): Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Springer.

Bradley, K & Smyth, B. (2003): Personalized information ordering: a case study in online recruitment. Knowledge-Based Systems, vol.16(5-6): 269-275

Bridge, D & Ferguson, A. (2002): An expressive query language for product recommender systems. Artificial Intelligence Review, vol. 18(3–4), 269–307.

Bridge, D., Göker, M., McGinty, L., & Smyth, B. (2006): Case-based recommender systems. *The Knowledge Engineering Review*, vol. 23(3): 315-320.

Burke, R. (2002): Interactive Critiquing for catalog navigation in e-commerce, vol.18(3-4): 245-267.

Díaz-Agudo, B. & González Calero, P. (2002): CBROnto: A Task/Method Ontology for CBR. Procs. of the 15th International FLAIRS 2002 Conference, pp.101-105, AAAI Press.

Herlocker, J., Konstan, J.A., Borchers, A. & Riedl, J (1999): An algorithmic frmaework for performing collaborative filtering. In Procs. of SIGIR, ACM Press, pp.230-237

McSherry, D. (2002): Diversity-conscious retrieval. In S.Craw & Preece, A (eds) Procs. of the 6th European Conference on Case-Based Reasoning. Springer, pp.219–233.

McSherry, D, (2003): Similarity and compromise. In K.D.Ashley & D.G.Bridge (eds) Procs. of the 5th International Conference on Case-Based Reasoning. Springer, pp.291–305.

Recio, J.A., Sánchez, A., Díaz-Agudo, B., & González-Calero, P.A. (2006): Lessons Learnt in the Development of a CBR Framework. In M.Petridis, (ed.): Procs. of the 11th UK Workshop on Case-Based Reasoning, CMS Press, University of Greenwich, UK.

Sarwar, B., Karypis, G., Konstan, J.A. & Riedl, J. (2001): Item-based collaborative filtering recommendation algorithms. In Procs of the 10th International Conference on the WWW, ACM Press, pp.285-295.

Smyth, B & McClave, P. (2001): Similarity vs. diversity. In D.W.Aha & I.Watson (eds) Procs of the 4th International Conference on Case-Based Reasoning. Springer, pp. 347–361.

Wilke, W., Lenz, M. & Wess.S (1998): Intelligent sales support with CBR. In M.Lenz et al. Case-Based Reasoning Technology: From Foundations to Applications, Springer, pp.91-113.