Reproducibility of CBR applications in COLIBRI [⋆]

Juan A. Recio-García, Belén Díaz-Agudo, and Pedro González-Calero

Department of Software Engineering and Artificial Intelligence
Universidad Complutense de Madrid, Spain
email: jareciog@fdi.ucm.es, belend@sip.ucm.es, pedro@sip.ucm.es

Abstract. There is an increasing requirement in the scientific software development area of promoting the interchange of resources to ensure the reproducibility and validation of the results. This paper presents the COLIBRI STUDIO environment that supports researchers in the generation of Case-based Reasoning (CBR) systems by means of workflow-like representations with different degrees of abstraction. These workflows –called *templates*– can be shared with the community to promote their future reference and reproducibility.

1 Introduction

COLIBRI is a platform for developing Case-Based Reasoning (CBR) software. Its main goal is to provide the infrastructure required to develop new CBR systems and its associated software components. COLIBRI is designed to offer a collaborative environment where users could share their efforts in implementing CBR applications. It is an open platform where users can contribute with different designs or components that will be reused by other users. In general terms, this process -named the COLIBRI development process- proposes and promotes the collaboration among independent entities (research groups, educational institutions, companies) involved in the CBR field. To enable this collaboration, the development process defines several activities to interchange, publish, retrieve, instantiate and deploy workflows that conceptualize CBR systems.

The first advantage of our proposal is the reduction of the development cost through the reuse of existing templates and components. This is one of the aspirations of the software industry: that software development advances, at least in part, through a process of reusing components. In this scenario, the problem consists of composing several software components to obtain a system with a certain behaviour. To perform this composition it is possible to take advantage of previously developed systems. This process has obvious parallels with the CBR cycle consisting on retrieval, reuse, revise and retain. The expected benefits are improvements in programmer productivity and in software quality.

Our COLIBRI development process [1] has an additional advantage that is analysed in this paper: the collaboration among users promotes the repeatability of the results achieved by other researchers. Nowadays the reliability of

[⋆] Supported by Spanish Ministry of Science and Education (TIN2009-13692-C03-03) the experimental results must be backed up by the reproducibility of the experiments. This feature ensures the advance in a research area because further experiments can be easily run by extending the existing ones. We propose a development process that promotes the reproducibility of experiments for the Case-based Reasoning realm.

The paper runs as follows: Section 2 introduces reproducibility and why it is relevant in the CBR realm. Section 3 presents the COLIBRI platform and its associated development process based on templates. Section 4 presents the tools that support reproducibility in COLIBRI and some issues associated to this process. Finally, Section 5 concludes the paper.

2 Reproducibility for the Case-based Reasoning realm

Reproducibility is a key element in the scientific method and enables scientists to evaluate the validity of each other's hypothesis. In general terms, scientific experiments are often done manually and are prone to error, slowing the pace of discoveries. The works by Gil et al. [2–4] analyse this problem and propose workflows technology as a suitable solution to aid researchers in the publication, discovery and reuse of existing computational processes. Workflows capture processes in a declarative manner so that they can be easily reproduced by other groups or replicated on other datasets.

Reproducibility is a requirement for any scientific research domain, and CBR is not alien to it. Advances in CBR are performed by several research groups that are continuously proposing novel techniques. These techniques must be compared to the existing ones to validate their correctness. However, the development of CBR systems is not an obvious task and requires a significant knowledge engineering effort and different skills on software development. This is a limitation that slows down the development of the CBR realm. Our working hypothesis is that Case-based Reasoning requires tools and procedures to support the scientific method like any other domain does. CBR methodologies and implementations should be appropriately indexed and made available for referencing and reuse. Benefits are manifold: automation of system generation, systematic exploration of the CBR design space, validation support, optimization and correct reproducibility are the most significant. Additionally, such capabilities have enormous advantages for educational purposes.

These goals have been achieved by existing computational workflow systems for generic computation [2]. Taverna is a workflow approach for the integration of components in the bioinformatics field [5]. It hides the complexity of the access to the processing services. Pegasus is another widely used worflow system that enables the composition of distributed resources [6]. However, to assist scientists in the composition, management and execution of these workflows, authors provide a complementary environment named Wings [7]. This tool starts with high-level user description and uses knowledge about components, data and workflows to automatically generate the workflows for Pegasus [8]. The Wing system points out the central role of a knowledge rich representation of workflows and their components, that is provided by means of ontologies. These ontologies are formalized in the OWL language [9], and the associated Description Logics reasoners [10, 11] provide the reasoning capabilities to ensure the correctness of the workflow and the compatibility with the data being processed. Moreover, Wings defines different layers of abstraction in the specification of workflows. These different specifications must be taken into account in order to provide the appropriate tools for the users. The highest layers of abstractions define the overall behaviour of the system whereas the lower layers include computational and execution details.

These contrasted ideas for the automatic reuse of workflows in generic scientific domains have being applied to the CBR realm in the COLIBRI platform. COLIBRI defines an architecture, development process and tools that provide the benefits of the workflows technologies previously described: reproducibility, automatic generation, and evaluation. We present this platform next.

3 The COLIBRI platform

Addressing the task of developing a CBR system raises many design questions: how are cases represented? Where is the case base stored and how are the cases loaded? How should algorithms access the information inside cases? How is the background knowledge included? and so on. A successful approach to solve these issues is to turn to the expertise obtained from previous developments. Therefore, COLIBRI proposes an architecture that states how to design CBR systems and their composing elements. The definition of this architecture is the key element of the platform as it enables the compatibility and reuse of components and worflows created by independent sources.

The main items of the COLIBRI architecture are:

– Persistence: defines how to organize the storing and loading of cases from different media like data bases, textual files, etc. It proposes the use of specialized connectors that perform this task.
– Definition of a clear and common structure for the basic knowledge models found in a CBR application: cases, queries, connectors, similarity metrics, case-base organizations, etc.
– Behaviour of the CBR systems: COLIBRI organizes CBR systems into: precycle, where the required knowledge models (mostly cases) are initialized; cycle, which performs the 4 R's tasks; and postcycle, where resources are released.
– Methods: They are software components that implement the different algorithms involved in the retrieval/reuse/revise/retain cycle. The platform also includes methods for maintenance and evaluation.

Any architecture will not be useful without a reference implementation that enables users to create tangible applications. The main features of this implementation should be reusability and extensibility to let users adapt it to the target system. In the COLIBRI platform this building block is provided by the jCOLIBRI framework. jCOLIBRI is a mature software framework for developing CBR systems in Java that has evolved over time, building on several years of experience[1] [12].

After jCOLIBRI was sufficiently mature we have continued with the next step in our platform: the graphical development tools to aid users in the development of CBR systems through the reuse of existing designs. These tools are enclosed in the COLIBRI Studio IDE[2] and are described in detail in the next section.

The incorporation of such tools into the platform has required the definition of a software development process that identifies the task required to implement CBR systems and the different user roles involved in this process. The following Section describes this software development process, its activities, the associated user roles and the tools that support it.

3.1 The COLIBRI Development Process

The main feature of the development process proposed in COLIBRI is reuse (somehow inherent to CBR). We propose the reuse of both: system designs and their components. Reusable designs are called *templates* and comprise CBR system workflows which specify the behaviour of a set of CBR systems. In general terms, the platform provides a catalogue of templates and lets users select the most suitable one and adapt it to the concrete requirements of the target application. It is a kind of CBR process for developing CBR systems [13].

The tools provided by our platform COLIBRI are targeted to different user roles. We have identified several roles that face the development of CBR systems from different points of view: senior researchers design the behaviour of the application and define the algorithms that will be implemented to create software components that are composed in order to assemble the final system. On the other hand, developers will implement these systems/components. Furthermore, during the last few years there has been an increasing interest in using jCOLIBRI as a teaching tool, and consequently, our platform also eases this task.

User roles are associated to activities that conform the COLIBRI development process. Next, we describe these activities and the user roles that perform them:

Template generation. A template is a workflow-based representation of a CBR system where several tasks are linked together to define the desired behaviour. They should be generated by 'expert' users although other users may create them. Here reputation will play a significant role as we will discuss in Section 4.1. This is the first activity in our software development process. Figure 1 shows a basic template in our specialized tool to design templates.

[1] Download, reference and academic publications can be fount at the web site: www.jcolibri.net

[2] Available at: www.colibricbrstudio.net

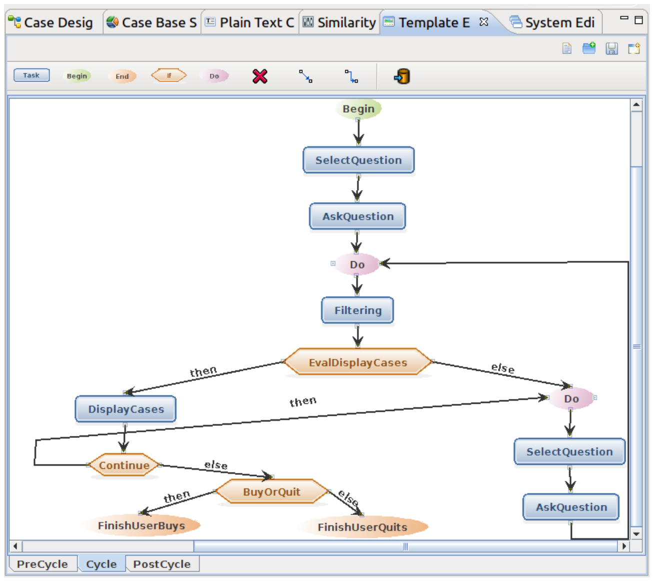
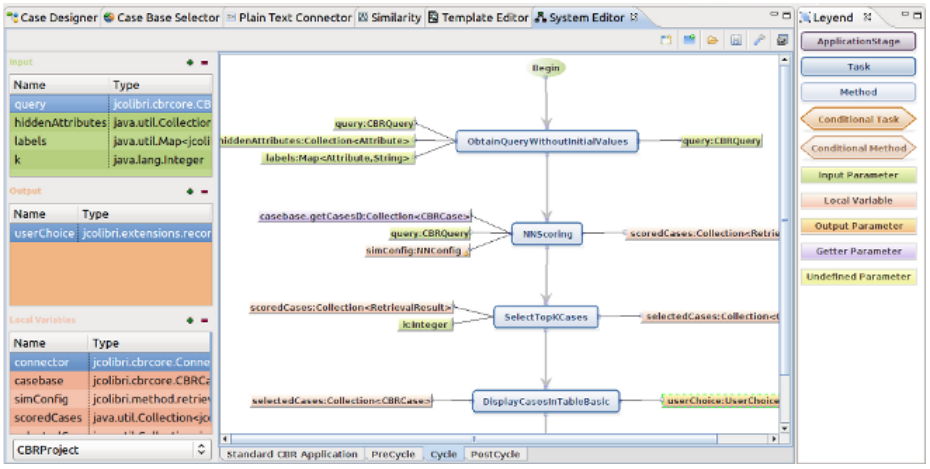


Fig. 1. Screenshot of the template generation tool.

Template publishing. Templates can be shared with the community. Therefore there is a second tool that lets users publish a template in the COLIBRI central repository.

Template retrieval and adaptation. Although the publication of templates is a key element in the platform, the main use case consists of retrieving and adapting the template to generate a new CBR system. Here the actors are not only CBR experts: developers, teachers, students, or inexperienced researchers will perform these activities. Due to their importance, these activities are referred to as the Template Base Design (TBD). The TBD begins with the retrieval of the template to be adapted from the central repository. This retrieval is performed by means of a recommender system proposes the most suitable template depending on the features of the target CBR system. It follows a "navigation by proposing" approach where templates are suggested to the user. Next, the adaptation of the template retrieved consists of assigning components that solve each task (see Figure 2). These components are the ones provided by the jCOLIBRI framework. For a detailed description of the TBD we point readers to [13] although we include a discussion about the implications of these activities from the reproducibility point of view in Section 4.1.

Component development. The design of components is closely related to the advance in CBR research as they implement the different algorithms being discovered by the community. Therefore, this is the second main task of expert researchers. However, we do not expect that expert researchers



Name	Type
query	jcolibri.cbrcore.CB
hiddenAttributes	java.util.Collection
labels	java.util.Map<jcoli
k	java.lang.Integer

Name	Type
userChoice	jcolibri.extensions.recor

Name	Type
connector	jcolibri.cbrcore.Conne
casebase	jcolibri.cbrcore.CBRCa
simConfig	jcolibri.method.retriev
scoredCases	java.util.Collection<jc

Fig. 2. Screenshot of the template adaptation tool.

will implement the components. This task will be delegated to users in a 'developer' role. We also contemplate the role of 'junior researcher' that could design and even implement his own experimental components. Again, these components could be shared with the community by means of the publication tool that uploads it to the COLIBRI repository.

System Evaluation. As we have mentioned, one of the most relevant benefits of COLIBRI is that it provides an easy to use experimental environment to test new templates and components. Consequently another activity in the development process is the evaluation of the generated systems. It enables the comparison of the performance of different CBR system implementations.

Up to now, we have referred to templates as a general term. Next we will provide more details about their formalization and conceptualization.

3.2 Template Categorization

Researchers may have different points of view regarding the data to be published and shared with the community. Some researchers may be keen on sharing all the details of the experiment, whereas others may desire to publish only an abstract description. This fact must be taken into account in a collaborative environment like COLIBRI Studio. The level of abstraction has an impact in the reproducibility of the CBR system as abstract templates do not provide enough details to generate the original system. However, the tools in COLIBRI Studio support the instantiation of this abstract representations to end up with executable systems.

Consequently, the COLIBRI development process defines the following categorization of templates according to their level of abstraction:

Abstract templates. Abstract templates are a high level representations of CBR systems. This kind of templates comprises CBR system designs which specify behaviour but they do not explicitly define functional details by means of tasks. A task defines, in an independent way, a piece of functionality that must be provided by the system. Tasks conform a workflow that states the execution (control-flow) of the application. Figure 1 shows an abstract template in the template editor tool in COLIBRI Studio.

Instantiated templates. The functionality specified by tasks can be provided by different software components. This way each task in a template can be *solved* by a component that implements the expected behaviour. When every task in a template has a component assigned –usually from the jCOLIBRI framework–, that template is considered *instantiated* with a concrete configuration of reasoning algorithms that implement a particular CBR application. The instantiation activity is supported by the adaptation tool of COLIBRI Studio. This activity implies the connection between components' inputs and outputs to define the data-flow of the system. Figure 2 shows an instantiated template generated from an abstract template through the template adaptation tool.

Executable Systems. An instantiated template encapsulates the algorithmic steps that perform the reasoning cycle of a CBR application. However, a fully functional CBR system requires additional knowledge models such as the case base or the retrieval (similarity) and reuse knowledge. These models can be defined by means of the tools in COLIBRI Studio. When an instantiated template is configured with these knowledge models, we obtain a runnable or executable system. COLIBRI Studio is able to generate executable systems automatically because the control flow of the application is defined by the initial "abstract template", and the concrete algorithmic details are specified during the data flow configuration of the "instantiation process".

Once we have identified the abstraction levels in the representation of templates, we can describe how to exploit these representations to support reproducibility of CBR applications.

4 Reproducibility in COLIBRI Studio

COLIBRI Studio supports the publishing of templates, components and data in a central repository that is later explored by the user to retrieve and reuse these existing resources (i.e. the Template-based Design). TBD enables the reproduction of existing systems and their further extension or evaluation. However, the nature of the element being published has an impact in the reproducibility of the system. Next we analyse these dependencies.

- At the lowest reproducibility level we find abstract templates. Users reusing abstract templates have the support of COLIBRI Studio to instantiate them with the components from jCOLIBRI. However the executable system obtained may not be identical to the original one as the algorithms chosen to solve each task may be different. Furthermore, the knowledge containers (case base, similarity, etc.) are not provided and must be supplied by the user reusing the template.

- Instantiated templates, provide a higher reproducibility level than abstract ones. They specify the components used to solve each task but do not include the knowledge containers. Note that those components could be custom implementations created by the developers of the original system/template. If an instantiated template includes such a components, they should be also published to ensure the reproducibility.
- Executable systems. They are systems completely reproducible by other users. They include all the required elements.
- Components. System developers can also publish components that provide the behaviour defined by a task. These components may be just just new algorithms made available to the community to solve common tasks, or key components for the instantiation of custom templates published in the repository by the same authors.
- Knowledge models. This is the last element of a CBR system that can be shared both as part of a executable system, or independently. COLIBRI STUDIO supports the publishing of four types of knowledge containers:
  - Case Structure. Defines the typed attributes that conform a case. They are created through one of the tools in COLIBRI STUDIO.
  - Case Base. A case base is stored in a persistance media (data base, text file, ontology, ...) and loaded through a connector into the working memory of cases. Connectors are components provided by the jCOLIBRI framework or created ad-hoc by developers. They are configured with xml files that define how cases are mapped from the persistance media into a concrete case structure. These elements conform the case base knowledge model: persistance, connector and configuration. They can be packaged together and shared with the community.
  - In-memory organization. Once cases are loaded from persistance (through a connector) the are indexed for being used by the components that instantiate a template. There are several in-memory case base organizations such as linear lists or k-d trees. Developers can also implement their own organizations.
  - Similarity knowledge. Similarity is defined in COLIBRI by means of global and local similarity functions assigned to each attribute that conforms the case structure. A tool in COLIBRI STUDIO allows users to define the configuration of the similarity metrics. It is the last knowledge model of a CBR system that can be shared with the community.
- Evaluation metrics. COLIBRI includes a complete framework for the evaluation of CBR systems. It includes cross-validation techniques and different performance metrics. Although it is an optional requirement, the evaluation configuration can be also published to provide a complete reproducibility of the experimental results.

4.1 Reputation and Provenance

Once we have presented the elements to be published and retrieved, several questions arise: Who should publish these elements? What impact does the publisher have in the retrieval of templates? As we mentioned in Section 3.1, templates should be generated by expert users. And this expertise should be taken into account during the retrieval process. It implies that the authorship reputation within the community should have an impact in the recommendation process that aids users in the retrieval of templates. Reputation can be hard-coded into the system, listing well-known researchers as trustable authors, or may appear, as described below, through collaborative recommendation. As users evaluate the contributions from other users, reputation will grow for those authors contributing templates that others like. Templates are not the only element that is influenced by this provenance knowledge; the retrieval of components, case bases, similarity knowledge, etc. should be also weighted according the reputation of the authors. Provenance is a raising topic associated to reproducibility [14, 15]. This factor must be also considered during the publication process and, consequently, provide the mechanisms to support the inclusion of the provenance knowledge. Although we have not included provenance mechanisms in our platform yet, we plan to implement collaborative approaches.

The current implementation of the recommender system for the retrieval of templates follows a content-based approach. It means that the requisites of the user are compared to the description of the templates to find the most suitable ones. However, instead of using the details of the template to perform the recommendation, it could be possible to follow a collaborative approach. Collaborative recommenders consider the opinions or ratings of the users to select the most suitable items. This is an alternative approach that could be implemented in the repository of templates to allow users to rate or comment templates, components, etc. This way, ratings and comments become a measure of the provenance or reputation of the author. Elements with higher valoration will be proposed first to users, and these users may later rate their experience. This rating process can be considered as the revision step of our CBR approach. Template-based design consists in retrieving and adapting templates. We plan to include this further stage where users provide revisions in a collaborative way.

The descriptions of templates and the other elements in the platform use a semantic representation that enhances the capabilities of the tools in COLIBRI Studio. This representation includes functional descriptions and must be also capable to integrate the provenance knowledge that we have described. Next we present the details of this semantic representation.

4.2 Semantic representation

As we have described previously, the COLIBRI development process involves several elements that must be integrated to obtain the target CBR system. Templates, components and knowledge models are combined by the user or the automatic tools provided by COLIBRI Studio. However, a major issue arises:

how does COLIBRI Studio control the semantic coherence of the system being generated? Methods are only able to solve certain tasks; tasks are thought to be performed over concrete case structures; and depending on the case structure developers need specific persistance connectors and similarity metrics. Additionally, reputation and provenance knowledge should also integrated in the development process. Therefore, there is a semantic interdependency between the elements involved in the development of a CBR system that must be controlled by COLIBRI Studio to support not only the local development of systems but also the publishing and reuse from the central repository.

The answer to this questions in the COLIBRI platform is the *CBROnto* ontology [16]. This ontology guides the description of components, the design of templates and their associated retrieval and adaptation activities. There are many formalisms for representing the templates such as UML. However we have decided to use a representation based on ontologies to enhance the whole development process. The most significant benefit of this representation is the reasoning capabilities. Several ontologies have been proposed for modelling systems. Most of them come from the field of Semantic Web Services (SWS). This community explores -among other things- the methodologies for automatically composing web services to create executable software systems. Each service is described by means of ontologies that define its semantic behaviour. Therefore, in the SWS community there are different standards to represent the behaviour of software components and their composition. The most significant examples are OWL-S [17] and WSMO [18].

Therefore, we have created our *CBROnto* ontology using the OWL language and integrated the vocabulary needed to represent templates (in their different categories), components and knowledge models. This ontology is used to describe each element in the COLIBRI platform and ensures their correct integration into executable systems. Conceptualizations in *CBROnto* include semantic restrictions that define the interdependencies between each element. These restrictions are represented using the Description Logics capabilities of the OWL language. Semantic restrictions can be used during the retrieval and adaptation of templates to guide the users. It requires the application of knowledge intensive similarity and adaptation techniques that take advantage of the semantic descriptions. For example, the recommender system that supports the retrieval of templates exploits this knowledge to find the most suitable template according to the user requirements. The similarity metric that compares templates to the requirements of the user exploits the semantic knowledge in *CBROnto* that conceptualizes the structure of the template, its associated components and several other high level descriptions provided by the author of the template.

Furthermore, the instantiation process that adapts templates (Figure 2) is guided by the ontological knowledge. *CBROnto* dictates the possible assignments of components to tasks, because it represents the functional requisites of tasks and the capabilities of the components. For example, in the Java signature of a component we can state that it requires a list of cases. However, without an additional mechanism such as *CBROnto* we could not define the number or nature of the cases contained (structured, textual, etc.). The reasoning capabilities of OWL enables the definition these restrictions for tasks and components. Consequently, our system is able to find the proper components that satisfy the semantic requirements of a task.

CBROnto shares many conceptual ideas with the Wings system described in Section 2. It includes an ontology that describes the elements in a workflow system that is exploited to validate the system being generated. Our approach differs in the way this ontology is exploited. In COLIBRI the knowledge rich representation of templates and components is used in a case-based fashion. Templates are retrieved and adapted following the typical CBR approach. This adaptation step is not supported in Wings. Somehow, we can define COLIBRI Studio as a knowledge-intensive CBR application where cases are knowledge-rich workflows: the templates [13].

The role of *CBROnto* in the COLIBRI development process is a large topic that cannot be exposed in this paper due to space restrictions. We point users to [19] for details. *CBROnto* still does not include the reputation and provenance knowledge required to implement a proper repository of templates. This is our on-going work.

5 Conclusions

In this paper we have presented our COLIBRI platform and its associated development process from the point of view of reproducibility. We propose the reuse of templates that can be shared with the community to promote their future reference and reproducibility. COLIBRI is a popular platform in the community. Its framework jCOLIBRI includes dozens of components for the implementation of CBR systems, many of them contributed by other research groups in the community. It has hit, as of this writing, the 10.000 downloads mark with users in 100 different countries. On the other hand, the configuration tools that supports the Template-based design have been recently published. It includes, by now, all the components from jCOLIBRI and up to 16 templates for the generation of recommender systems. We are working on the inclusion of templates for other CBR families such as textual CBR, web CBR or group recommendations. In case COLIBRI Studio would reach the same popularity than jCOLIBRI, reproducibility in CBR would be closer to become a reality.

References

1. Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A.: Template based design in colibri studio. In: Proceedings of the Process-oriented Case-Based Reasning Workshop at ICCBR'11. (2011) 101–110
2. Gil, Y.: From data to knowledge to discoveries: Artificial intelligence and scientific workflows. Scientific Programming **17** (2009) 231–246
3. Gil, Y., Deelman, E., Ellisman, M.H., Fahringer, T., Fox, G., Gannon, D., Goble, C.A., Livny, M., Moreau, L., Myers, J.: Examining the challenges of scientific workflows. IEEE Computer **40** (2007) 24–32

4. Gil, Y., González-Calero, P.A., Kim, J., Moody, J., Ratnakar, V.: A semantic framework for automatic generation of computational workflows using distributed data and component catalogues. J. Exp. Theor. Artif. Intell. **23** (2011) 389–467
5. Oinn, T., Greenwood, M., Addis, M., Alpdemir, N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M., Senger, M., Stevens, R., Wipat, A., Wroe, C.: Taverna: lessons in creating a workflow environment for the life sciences. Concurrency and Computation: Practice and Experience **18** (2006) 1067–1100
6. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A.C., Jacob, J.C., Katz, D.S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming **13** (2005) 219–237
7. Gil, Y., Ratnakar, V., Kim, J., González-Calero, P.A., Groth, P.T., Moody, J., Deelman, E.: Wings: Intelligent workflow-based design of computational experiments. IEEE Intelligent Systems **26** (2011) 62–72
8. Gil, Y., Ratnakar, V., Deelman, E., Mehta, G., Kim, J.: Wings for pegasus: Creating large-scale scientific applications using semantic representations of computational workflows. In: AAAI, AAAI Press (2007) 1767–1774
9. W3C: Owl web ontology language overview. http://www.w3.org/TR/owl-features/ (2004) World Wide Web Consortium.
10. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
11. Sirin, E., Parsia, B.: Pellet: An OWL DL Reasoner. In Haarslev, V., Möller, R., eds.: Description Logics. Volume 104 of CEUR Workshop Proceedings., CEUR-WS.org (2004)
12. Díaz-Agudo, B., González-Calero, P.A., Recio-García, J.A., Sánchez, A.: Building CBR systems with jCOLIBRI. Special Issue on Experimental Software and Toolkits of the Journal Science of Computer Programming **69** (2007) 68–75
13. Recio-García, J.A., Bridge, D.G., Díaz-Agudo, B., González-Calero, P.A.: CBR for CBR: A Case-Based Template Recommender System for Building Case-Based Systems. In: ECCBR08. Volume 5239 of LNCS., Springer (2008) 459–473
14. Gil, Y., Szekely, P.A., Villamizar, S., Harmon, T.C., Ratnakar, V., Gupta, S., Muslea, M., Silva, F., Knoblock, C.A.: Mind your metadata: Exploiting semantics for configuration, adaptation, and provenance in scientific workflows. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N.F., Blomqvist, E., eds.: International Semantic Web Conference (2). Volume 7032 of Lecture Notes in Computer Science., Springer (2011) 65–80
15. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P.T., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E.G., den Bussche, J.V.: The open provenance model core specification (v1.1). Future Generation Comp. Syst. **27** (2011) 743–756
16. Díaz-Agudo, B., González-Calero, P.A.: CBROnto: a task/method ontology for CBR. In Haller, S., Simmons, G., eds.: Procs. of the 15th International FLAIRS'02 Conference, AAAI Press (2002) 101–105
17. The OWL Services Coalition: OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1/overview/ (2004)
18. Cristina Feier, J.D.: Wsmo primer. http://www.wsmo.org/TR/d3/d3.1/v0.1/ (2005)
19. Recio-García, J.A., González-Calero, P.A., Díaz-Agudo, B.: Template-based design in colibri studio. Journal on Information Systems (To appear)