Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 50 (2004) 87–115
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

A case-based reasoning framework for workflow model management

Therani Madhusudan *, J. Leon Zhao, Byron Marshall

MIS Department, University of Arizona, Tucson, AZ 85721, USA

Available online 25 January 2004

Abstract

In order to support efficient workflow design, recent commercial workflow systems are providing templates of common business processes. These templates, called cases, can be modified individually or collectively into a new workflow to meet the business specification. However, little research has been done on how to manage workflow models, including issues such as model storage, model retrieval, model reuse and assembly. In this paper, we propose a novel framework to support workflow modeling and design by adapting workflow cases from a repository of process models. Our approach to workflow model management is based on a structured workflow lifecycle and leverages recent advances in model management and case-based reasoning techniques. Our contributions include a conceptual model of workflow cases, a similarity flooding algorithm for workflow case retrieval, and a domain-independent AI planning approach to workflow case composition. We illustrate the workflow model management framework with a prototype system called Case-Oriented Design Assistant for Workflow Modeling (CODAW).

Keywords: Case-oriented workflow modeling; Case-based reasoning; Ad hoc workflows; Model reuse

1. Introduction

Business process modeling is a critical activity in modern organizations to enable enterprise application integration, standardization of business processes, and online B2B and B2C E-commerce. The importance of business process modeling in IT-enabled business process management

* Corresponding author.
E-mail addresses: madhu@email.arizona.edu (T. Madhusudan), lzhao@bpa.arizona.edu (J.L. Zhao), byronm@eller.arizona.edu (B. Marshall).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2004.01.005

strategies is indicated by the recent calls for improving process modeling and process management in organizations [14].

Workflow modeling involves the translation of high-level business requirements into workflow schemas that can be executed by appropriate workflow engines. Specifying a workflow model is a knowledge intensive endeavor because development of a typical workflow model requires detailed understanding of the business process logic, the organizational chart, and the information systems accessed by the workflow. Further, a given informal business process description may be modeled in multiple ways, depending on the underlying IT infrastructure and business context.

Recent research has suggested the reuse of field tested process knowledge and associated process models to guide workflow modeling and design efforts [18,20,24]. Towards this end, recent commercial systems (such as Oracle Workflow–11*i*, INCOME [35], and ARIS [39]) provide basic templates for business processes, such as order processing and procurement. These templates may be instantiated and appropriately modified to an organization's needs by a knowledgeable workflow designer. However, standards for template representation and associated ontologies, formal guidelines for reuse of these templates, rules for their instantiation or modification, and procedures for their composition into complex workflows are currently non-existent. Furthermore, there is a lack of design guidelines and workflow modeling tools at present to support tasks such as generation of alternative workflow process models for a given set of business requirements [40].

Workflow modeling and design is the task of defining structured workflow schemas from informal business requirements that satisfy a variety of business logic, organizational and resource constraints. We note that the terms workflow schema, process model, and workflow model are used synonymously in this paper. Workflow modeling involves definition and selection of appropriate tasks (possibly from a task library), sequencing of the tasks to satisfy data and logical dependencies, allocation of resources consumed by the tasks, allocation of agents to execute tasks, scheduling of tasks considering concurrency, and finally, validating and verifying the model [1]. Manual workflow modeling is supported by graphical interfaces, where the workflow model is defined as a graph. Workflow modeling involves searching (albeit implicitly) through a design space defined by a large number of process model alternatives and selection of an optimal process model to fulfill the given problem. With the increasing adoption of workflow management systems and the advent of flexible process integration technologies such as web services, there is an acute need for developing tools and approaches to support workflow design and modeling.

In this paper, we propose a workflow model management framework based on a structured workflow design process, which enables reuse of process knowledge (both structured and unstructured) from organizational process repositories. The workflow design process consists of two phases. In the first phase, relevant business tasks are ordered into a seamless whole, satisfying pre-conditions and post-conditions. The result of this phase is a workflow model, a project network defined by a partial ordering amongst all the relevant tasks. Multiple workflow models may be designed to fulfill a given set of business goals. In the second phase, a process model is selected from the available alternatives and is further annotated with appropriate agents, resources and timing information, followed by incorporation of routing details, such as appropriate forks and joins to facilitate concurrent execution. Both phases of design may reuse process knowledge from available repositories.

We use a case-based reasoning (CBR) approach [22], which consists of case retrieval, case reuse, case adaptation and case verification tasks, to support workflow model reuse during workflow

design. CBR is a computational approach that supports explicit reuse of partial and possibly incomplete, experiential knowledge (stored as cases) in solving ill-structured and complex cognitive tasks, such as design. Past knowledge may be reused to explore the workflow design space and synthesize new solutions. The CBR-approach to workflow model management has been prototyped in a system called Case-Oriented Design Assistant for Workflow Modeling (CODAW), which currently supports the first phase of design mentioned above. Development of an effective case representation, similarity-based retrieval algorithms, and case composition techniques are essential steps in the development of a CBR system to support workflow modeling and design. The main contributions of our research are:

- The definition of a structured workflow case representation that includes both declarative and procedural descriptions. The procedural description is based on the process graph metamodel detailed in [3] and the declarative description is based on a predicate logic-based situational calculus formalism of Artificial Intelligence (AI) planning [38].
- The development of a similarity flooding for workflow (SFW) algorithm to support retrieval of procedural workflow models, based on inexact matching of graph queries. The algorithm is derived from the similarity flooding (SF) algorithm [30], recently proposed for metadata model management in database systems. It is based on identifying local structural similarities between two directed labeled graphs, namely, a query graph and the process graph model of a workflow schema.
- The development of a workflow composition procedure using the Hierarchical Task Network (HTN) AI planning technique [32,34]. CODAW uses the declarative representation (mentioned above) to compose workflow models consisting of both sequential and concurrent tasks that support fulfillment of business requirements encoded as goals involving transformation of an initial business state into a final goal state.

The remainder of the paper is organized as follows. In Section 2, we discuss the relevance of case-based reasoning for workflow modeling and the recent advances in model management and AI planning. Section 3 provides an overview of CODAW, including the representation for workflow cases and organization of the process repository. In Section 4, we present the procedures for case retrieval and composition. Discussion of the proposed approach and concluding remarks are given in Section 5.

2. Literature review

Our research builds on multiple streams of related research, including workflow modeling, case-based reasoning, design automation and AI planning. Research in workflow modeling has addressed model reuse related issues in the context of standardizing workflow model representations, support for workflow design and workflow model acquisition. The need for structured approaches to workflow modeling and developing workflow models for both buildtime and runtime inter-operability is outlined in [8,40]. Recent research has focused on developing process modeling languages [3] and extending workflow modeling to support on-the-fly process design [13]. In the context of workflow design and management, process model reuse from organizational

memories has been discussed in [43]. The proposed system supports retrieval of workflows based on a well-defined taxonomy. Modifications to retrieved cases are performed manually to a non-obvious case representation. A system called EULE discusses a knowledge representation framework to support modeling of business processes [37]. The need for case-based support for ad hoc workflow model design is discussed in [42]. A catalog of workflow template patterns for guiding manual design are listed in [2,12]. Manual retrieval and reuse of models from process repositories is supported in tools such as ARIS Designer [19] and INCOME [35]. These tools use a variety of process representations including process graphs, Event-Process-Chains and Petri Nets to store workflow models. Process model reuse in the context of process mining is discussed in [4]. In summary, workflow model reuse has been addressed in a variety of contexts. However, these systems use proprietary process representations and provide minimal support for intelligent process model retrieval and automated model composition.

The proposed workflow model management framework is based on recent advances in case-based reasoning techniques [22]. Case-based reasoning is a problem solving technique based on the hypothesis that reasoning is reminding. That is, problem solving utilizes past experiences. CBR systems have proven useful in domains with weak models and a large body of unstructured, experiential knowledge [5,23]. Successful CBR-based systems have been developed for supporting both product and process design [17,27,31,33]. A case-based approach to a process design activity such as workflow modeling is feasible because of the recurrence of similar business tasks in different contexts, existence of similar types of data and task dependencies, and recurrence of common types of business constraints across a variety of business processes. The generic CBR problem solving cycle (called CBR cycle) is illustrated in Fig. 1, and consists of the following steps [5]: *retrieval of relevant (similar) cases from the repository based on cues derived from problem requirements, reuse of applicable cases* to suggest solutions to a new problem, *knowledge-based revision of relevant cases, testing-based verification and rule-based validation* to ensure correctness, and *retention of past solutions and failures* to enable learning. A variety of application-specific case retrieval (during the reuse phase) and case adaptation (during the revise phase) techniques are discussed in [22]. However, many of these techniques are not directly applicable for retrieval of workflow models, represented as directed graphs, because of the need for a flexible notion of similarity that combines features of domain knowledge and process graph structures in a reliable manner. Appropriate computational techniques need to be developed for utilizing CBR for workflow design.

Exact and inexact graph isomorphism algorithms for case retrieval are discussed in [41]. Workflow model retrieval can benefit from recent advances in the field of *model management* in databases. Generic meta-model management operations such as match, difference, merge and compose are outlined in [6]. For example, schema matching in databases is a particular instance of the generic model matching task. The similarity flooding algorithm for inexact, similarity based schema matching is presented in [30]. The similarity flooding algorithm for workflow case retrieval is discussed in Section 4.

The reuse and revise phases of the CBR cycle may involve minor or major modifications (called case adaptation) of the retrieved solutions. Compositional adaptation of cases in CBR systems is called case-based planning. Case-based planning is based on domain-independent AI planning technologies. Background to AI planning is provided in [38] and case-based planning is discussed in [9]. The technology of case-based planning has been successfully used in various process design

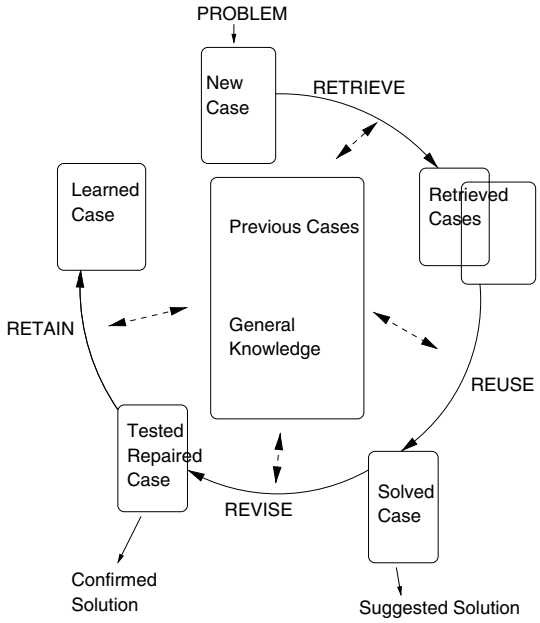

Fig. 1. The case-based reasoning cycle.

contexts such as manufacturing process planning, space mission scheduling and route planning [22]. Recent advances in the speed and performance of AI planning algorithms make it feasible to embed them in real-world applications [44]. In [25], the use of AI planning is suggested for software design at higher levels of abstraction, such as process modeling.

Recent research has also suggested the use of AI planning techniques for workflow management [10]. A declarative definition of a workflow model is equivalent to *a plan* to fulfill a business need. Case-based planning in our framework uses the Simple Hierarchical Ordered Planning (SHOP) algorithm, an implementation of the Hierarchical Task Network planning technique [34]. The SHOP algorithm supports reasoning about interactions between task pre-conditions and post-conditions during state-space search for developing plans. Additionally, SHOP allows reuse of appropriate prototypical and instance-level cases from repositories during problem solving [32]. With this background, we discuss the features of our proposed framework in the subsequent sections.

3. Overview of CODAW

The structured workflow design process, supported by the Case-Oriented Design Assistant for Workflow modeling, is shown in Fig. 2. The design process supports the incremental refinement of a workflow specification and is partitioned into the four phases of the CBR cycle (in Fig. 1), namely, retrieval, reuse, revise and retain.

The repository stores workflow schemas (called prototypical cases) and workflow instances (called instance-level cases). Note that we use the term *case* to refer to a unit of knowledge in the

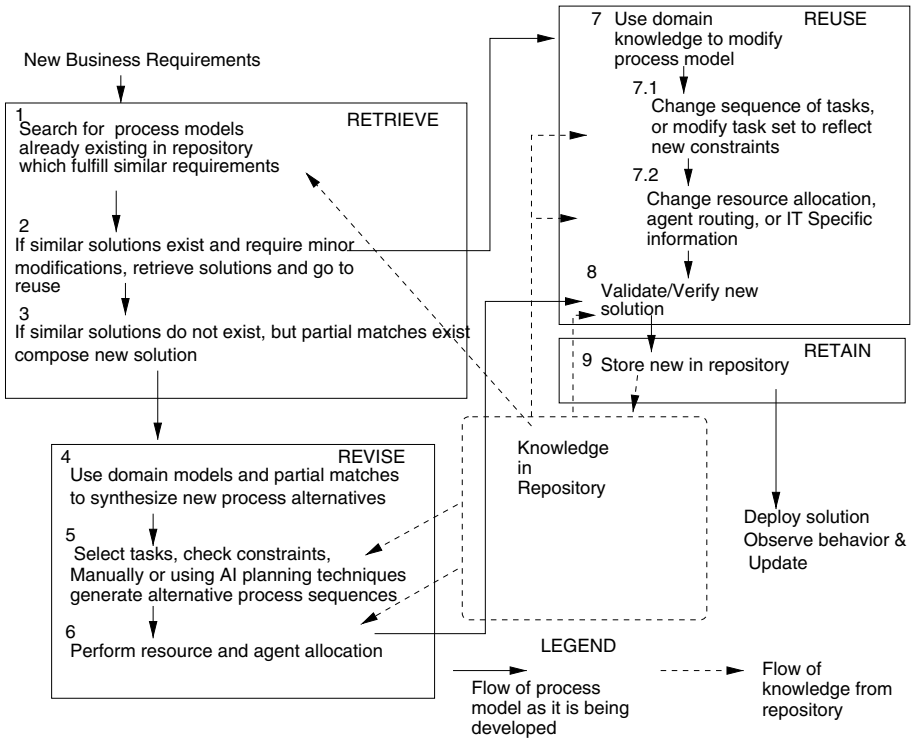

Fig. 2. Structured workflow design cycle.

repository as is done in the CBR research community. Cases contain both structured and unstructured information from all stages of the workflow modeling lifecycle such as business specification, formal modeling and encoding, validation and execution.

The design process in Fig. 2 (flow shown by the solid arrow, numbers indicate steps) begins when new business requirements are provided and used to initiate a search of the case repository (Step 1). Appropriate cases from the repository may be retrieved using text query mechanisms for unstructured textual data, Extensible Markup Language (XML) based query mechanisms for semistructured data, and domain specific mechanisms for proprietary data structures. The retrieval phase ensures that available knowledge, which may be useful to model the new business requirement, is identified upfront during the design process. Retrieved cases are analyzed (primarily manually, possibly automated) to decide if any of the retrieved knowledge, particularly process models, is appropriate for further modification or a new solution needs to be generated from scratch (Steps 2 and 3).

During the reuse phase, a retrieved process model (selected from possibly multiple alternatives) may be modified (Step 7). Appropriate domain knowledge may be used to alter the process sequence, new tasks may be added or deleted and constraints may be reconfigured (Step 7.1). Following task sequencing, resource allocation issues may be considered to enable concurrent execution and infrastructural specifications such as data location and agents added to the workflow model (Step 7.2). This refined model is then subject to validation and verification based on domain dependent rules, animation, simulation and formal approaches (Step 8). Successful validation may

trigger deployment in the Workflow Management System (WFMS). Additionally, this newly developed solution is stored in the repository during the retain phase of the design cycle (Step 9).

Alternatively, synthesis of a completely new solution may be chosen and initiated in the post retrieval analysis step (Step 4). Two alternative scenarios are possible in the revise phase of the CBR cycle. In the first scenario, partially matched process models from the retrieval step may be adapted to develop a new workflow. This step is commonly referred to as *customization* in workflow modeling. In the second scenario, no partial matches may be returned from the retrieval phase and a new workflow model may need to be synthesized from scratch, based on first principles domain knowledge. In CODAW, we support both scenarios using case-based planning. Each workflow schema (a prototypical case) is modeled as a composite task consisting of primitive tasks available in a pre-defined task library. In the first scenario, a new workflow may be composed from composite tasks, whereas in the second scenario, a new workflow may be synthesized by composition of primitive tasks. Both types of composition are supported by case-based planning, described in Section 4.2. Steps 4 and 5 may lead to multiple workflow model alternatives, which are then detailed with scheduling and resource information (Step 6). A workflow model (selected from the alternatives based on criteria such as estimated cost, resource usage and performance) is verified and then validated (Step 8).

It is important to note that each of these design steps relies on different snippets of knowledge resident in the process knowledge repository (whose use is illustrated by the dashed arrows). Successful completion of the design steps results in a workflow schema that fulfills the requirements, which may then be executed by a WFMS. We observe that each step of the design cycle may be executed manually or supported by an automated design system. Further, execution of the complete cycle may interleave manual and automated execution. Automated support is currently provided in CODAW for Steps 1, 2, 3, 4, and 5. Steps 6, 7, 8, and 9 are performed manually. Automated support for the above steps requires: (1) development of a case representation for workflows and initialization of the case repository with appropriate cases, (2) development of case retrieval algorithms based on exact and inexact (similarity) based matching, and (3) development of a planning-based technique for composition of retrieved cases.

In the following sections, we describe the case representation and retrieval algorithms in the context of a common workflow design scenario, namely, design of new product development (NPD) workflows [36]. With the short product life-cycles and competitive markets in the current business environment, many organizations are developing workflow models for managing the NPD process effectively to reduce overall product costs.

3.1. Case representation

Two types of cases are stored in the CODAW repository. Workflow schemas are stored as *prototypical cases*. Each instantiation of a prototypical case (a case in workflow terminology) is stored as an *instance-level* case. Prototypical cases embed the overall sequence of activities that fulfill a generic business requirement. Instance-level cases are execution traces of prototypical cases for well-specified inputs. Each prototypical case may have one or more instance-level cases associated with it. When a workflow schema is modified or updated, it is stored as a new prototypical case in the repository. Prototypical and instance-level cases are represented in CODAW with well-defined syntactic and semantic elements which are discussed below.

Our repository development efforts have focused on acquiring workflow models in the areas of engineering design, product development and supply chain management. We have developed the appropriate ontology based on current business practices in these functional areas, available template descriptions for these functions in commercial workflow tools, previous research on developing process repositories at higher levels of abstraction, such as process handbooks [28], and current efforts on developing ontologies for supporting the Semantic Web [11,16]. We describe the ontology for NPD as needed in the ensuing discussions. In our process ontology, we assume the existence of primitive tasks in a given business domain (as indicated earlier). These primitive tasks may be combined into complex workflows. A workflow schema defines the internal structure of a composite task, which may then be reused as component task in the design of additional workflows. Consider the development of a case representation for the workflow schemas shown as Unified Modeling Language (UML) activity diagrams in Fig. 3. The UML activity diagram is only used for illustrative purposes and shows the activities and control flow including concurrency nodes (such as forks and joins) for the schemas. Fig. 3A is a nominal NPD workflow schema. Shown at the top of the figure are the different organizational roles (swimlanes) that are involved in executing the activity (or task). Activities may be performed by a single agent or teams of agents. The span of the activity boxes denotes the various roles involved in performing the activity. For example, `Project Selection` and `Project Plan Review` are performed by all roles. Each of the activities could be represented by a primitive or composite task. Fig. 3B is the schema for a procurement process used to source design components from suppliers. This workflow is used to source a variety of engineered components and materials via a bidding and

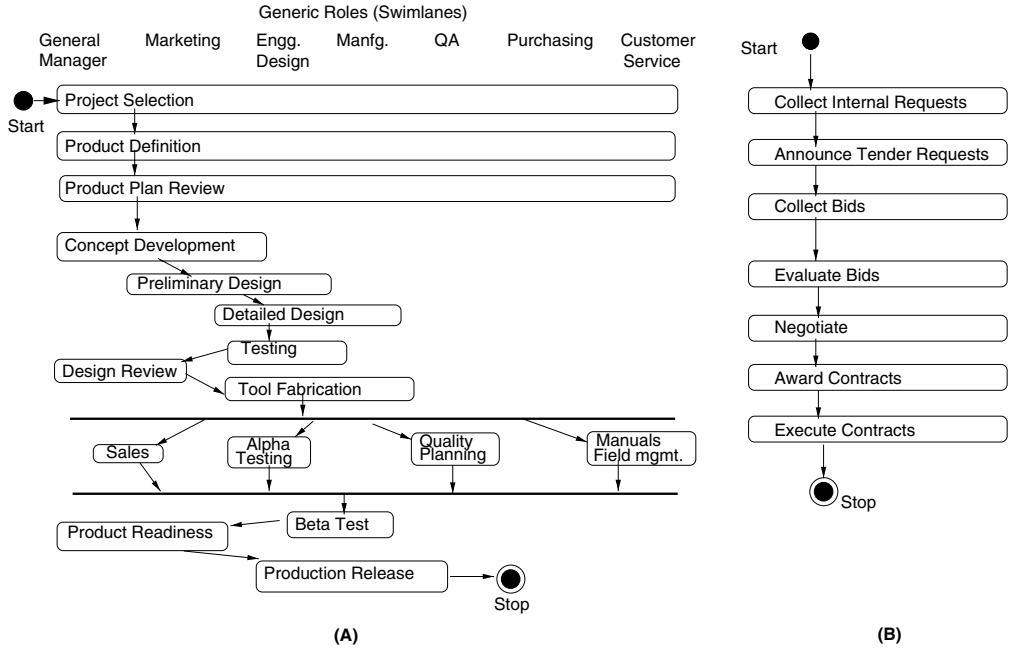

Fig. 3. Examples of workflow schemas: (A) product development and (B) procurement.

contracting process. In this process, offers from vendors, based on tender requests, are selected based on a variety of criteria.

Both these workflow schemas are stored as prototypical cases in the CODAW repository. Prototypical and instance-level cases are encoded in XML with appropriate tags for various structural elements. Details of the structural elements, corresponding tags and example XML representations are in Appendix A. Details of the structural elements of a prototypical case (a workflow schema) is listed in the first table. The elements of an instance-level case (a workflow instance) is listed in the third table. Table columns identify the element and its corresponding XML tags, describe the semantics of the element and illustrate its computational representation. Note that a single structural element may be described by multiple XML tags. We use standard object and data modeling techniques for all the elements. The XML representation of the prototypical case corresponding to the NPD schema of Fig. 3A is listed in Appendix B. The case representation (encoded in XML) has been designed with a focus on extensibility and supports integration of declarative and procedural process models. We have adopted tags from standardization efforts such as XPDL, WSFL and XLANG and plan to develop appropriate XSLT based inter-conversion mechanisms between the CODAW case representation and standards (as shown in [29]).

Prototypical cases include a procedural process graph model, denoted by the `WSPGModel` tag. This process graph model is represented by the metamodel presented in [3]. The `WSPGModel` includes subtags for the list of task nodes, decision nodes (with type), concurrency nodes (fork, join) and a list of directed control flow edges. The XML encoding shows a partial representation for the NPD workflow. The declarative first order predicate logic-based representation utilizes the AI planning representation for composite (workflows) and primitive tasks [34,38]. The elements `WSPreconds` and `WSPostconds` for the prototypical case define the pre-conditions and post-conditions that ensure that the case can be treated as a composite task and used as a single unit. `WSInputs` and `WSOutputs` define the input and output parameters (such as a design problem and a budget) necessary to enact the case. The prototypical case also defines the types of resources that may be needed to execute the case (defined by `ResTypeList`). The task list identifies the primitive and composite tasks (other workflows) that define the workflow schema. Structural elements of a primitive task are defined in the second table of Appendix A. The example representation illustrates the task `Project_Selection`. This task can only be executed when a list of projects and resources is available illustrated by the predicates in `TPreCondns` and upon completion, the task has the effect of adding new projects (denoted by `TPostConds`). The input and output parameters of the task are denoted `TParams` and the actual procedural code that implements this task is defined by `TaskImpl`. In the example in Appendix B, the task, `Project_Selection` is executed manually by a group. Prototypical cases get instantiated for different inputs, which are stored as instance-level cases whose structure is described in the third table of Appendix A. The tag `WSInstances` lists all the different instance-level cases associated with the schema.

A corresponding instance-level case (for the above example) is listed in Appendix C. Instance-level cases are grounded versions of the prototypical cases. Tags `WIInitState` and `WIFinalState` reflect the actual values of the parameters of variables in the predicates that define the pre-conditions and post-conditions of prototypical cases (composite tasks) or primitive tasks. A composite or primitive task is applicable in a given state, when predicate variables can be *logically unified* with those in `WIInitState`. In Appendix B, the predicate

(available_new_problem ?designprob) in tag WSPreConds is unified with the predicate (allocated_problem HAstarter) in tag WIInitState of the instance case in Appendix C, when the logical variable ?designproblem takes on the value HAstarter. Satisfying all predicates in this manner ensures that case WS2 can be applied to the current state. Successful enactment of the task transforms the initial state into the final state, where the predicates of WSPostConds are added to or deleted from the current state. The declarative representation supports planning-based composition during which cases are retrieved using logical unification. The Preference_ranking element in the prototypical case enables ranking amongst multiple cases that may apply to a given situation. In summary, the process graph element of the representation supports similarity-based retrieval and the declarative representation supports case composition. Additional structural elements of instance-level cases include audit and monitoring elements such as usage histories, events and exceptions, whose basic structure is shown in the examples in the Appendix. Cases may be retrieved using this information via text or XQuery-based retrieval. Further research is required to leverage such audit information to guide workflow modeling.

3.2. Repository management

Case development and population of the case base is a key task in development of the CODAW system. Workflow cases to populate the repository have been derived from best practices handbooks for different functional areas, observations of manual process execution and process elicit- ation from knowledge workers. In the initial development of CODAW, we have populated our repository with cases acquired from real world projects in courses on systems analysis and workflow modeling. Additionally, we have also collected workflow cases from the research liter- ature and those bundled with available workflow software. The repository in a CBR system needs to be bootstrapped with an initial population of cases, before it can support the CBR cycle (Fig. 2). To develop this initial population of the case base, we are currently manually encoding our acquired workflows by developing the associated declarative and procedural work representations. Currently we have 60 prototypical cases of various organizational business processes in our repository along with multiple execution instances for each. Further, the repository also stores a corresponding collection of primitive tasks in the library.

Currently the set of prototypical and instance-level cases is organized as flat XML files in a directory hierarchy defined by the indexing structure shown in Fig. 4. The hierarchies based on functional area, task and organizational structure provide multiple indices into the case base and enable structured case management. The hierarchies were developed by analyzing the current cases in our repository. The functional hierarchy is oriented towards supporting cross-domain usage of cases such as a prototype case in supply chain being composed with a case in CRM. The organizational hierarchy is oriented towards enabling retrieval based on organizational structure. The task hierarchy is being developed to identify common tasks that may be performed in different functional areas. For example, the search for a case that includes a task described by the keywords "tender generation for bidding", may retrieve bidding-related tasks across multiple functional areas. An entry for the tender generation task in the task hierarchy supports retrieval of appropriate process models from different functional areas. However, for

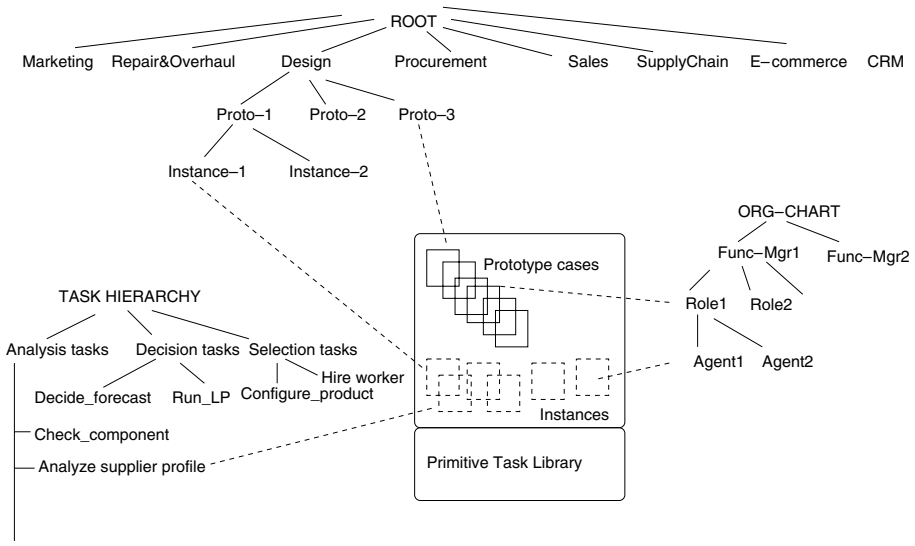

Fig. 4. Case organization hierarchy.

each type of bidding task, task parameters, inputs, outputs, preconditions and postconditions in each context may be different.

The indexing hierarchy supports the development of efficient domain-specific query mechanisms. As the number of cases increases, effective indexing is essential. Currently, during case retrieval, the indexing scheme is primarily used for guiding search (text retrieval or XQuery based) for a case across domains and also for effective filtering when there are large numbers of prototypical and instance cases. Text retrieval is supported using conventional term vector based information retrieval techniques. XQuery supports queries on the contents of the XML tags using exact matching. In the near future, we plan to extend the indexing scheme to include case organization by event-types, performance, and usage. Development of the current repository has focused on representational issues to support retrieval and composition, rather than minimizing space requirements and improving efficiency.

3.3. A scenario of workflow modeling with CODAW

To illustrate the use of CODAW for workflow modeling, consider that business needs evolve and changes are required to the NPD process of Fig. 3A. An organization (which is using this NPD workflow) is considering a strategic change wherein internal manufacturing is being reduced and upcoming product development efforts need to consider outsourcing of manufacturing. How should the product development workflow model (in Fig. 3A) be modified such that the tool fabrication step may be outsourced? Many alternatives may be possible. We outline a possible solution scenario below using CODAW.

Firstly, a workflow designer may search the repository for workflows related to outsourcing. The initial search may be performed using keywords (such as `Outsourcing`, `Procurement`) which may match relevant terms in the textual `Case Description` element of the case representation.

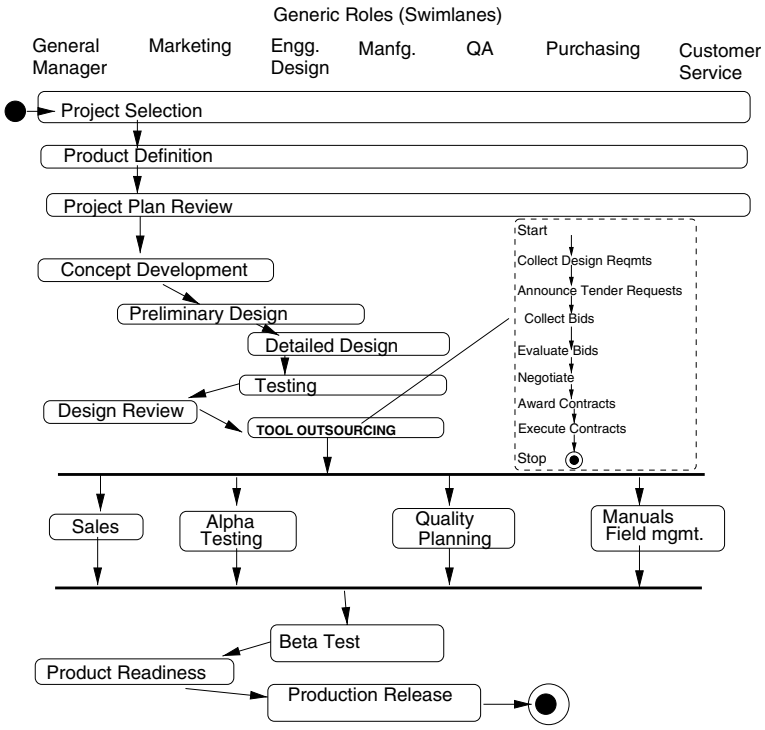

Fig. 5. A modified product development process.

The retrieved cases may include workflows wherein outsourcing is modeled as a primitive or composite task. One of the retrieved cases may be a procurement case (such as in Fig. 3B), which may then be selected and used to replace the `Tool Fabrication` task to develop a new process model as shown in Fig. 5. The instantiation of the composite task to replace the `Tool fabrication` task of the initial NPD process is performed manually. During this step, the designer may resolve pre-condition and post-condition requirements and modify tasks inside the composite task to ensure validity. Notice that task `Collect Internal Requests` in the original procurement process is replaced by the task `Collect Design Requirements` in the new process (shown in the dashed rectangular box in Fig. 5).

The development of the new NPD process illustrates how a particular type of outsourcing (via bidding) process may be retrieved from the repository and instantiated in the context of tool outsourcing. Additional design possibilities include: (a) the process repository may inherently contain a product development process model with outsourcing which may be retrieved and instantiated for the new product, (b) there may also be procurement processes wherein instead of tender-based outsourcing, auction mechanisms may be used, and (c) a new product development workflow may be composed by assembling one case (wherein the process model stops before tool design) with a procurement workflow case, wherein fabrication and followup is outsourced.

The NPD workflow design example illustrates the manual execution of the structured design process of Fig. 2. The process retrieval and adaptation phase of the design cycle provides a rich source of process modification possibilities (leading to a large design search space), in contrast to a

conventional approach to workflow design. Manually designing a product development workflow that handles each plausible task, resource and product combination for various business scenarios while capturing all the business rules and constraints is an overwhelming task. Automated support in CODAW would enable design steps such as recognizing and substituting tasks with appropriate workflows or composing individual workflow cases into new processes. Automated support for case retrieval and case composition is essential for effective use of a case-based design system. In the following section, we describe the similarity flooding algorithm for case retrieval and a planning-based approach for case composition.

4. Computational support for design in CODAW

The research objective in developing CODAW is to provide automated support for each step of the structured design process in Fig. 2. Fig. 6 illustrates the architecture of the CODAW prototype. Modules in CODAW support the different phases of the CBR cycle including retrieval, reuse, composition, adaptation and verification. Numbers in the figure indicate the different steps in supporting the CBR processes, which are coordinated by the `CBR Process Control` module. Case retrieval is initiated by a query (1), cues are generated from the problem description (2), and retrieval (3) is performed at the case manager, cases are ranked (6) and returned (7). Case addition may be supported by the indexing and repository update steps (4 and 5.1). Alternatively, retrieved cases may be adapted and verified (8 and 5.2) or multiple cases composed (8 and 5.3) and the

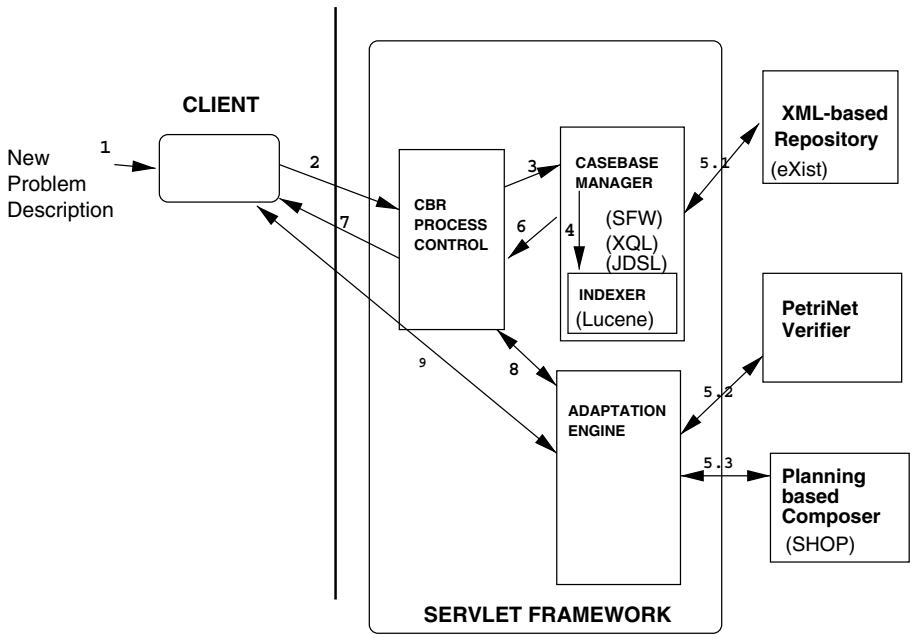

Fig. 6. Architecture of the CODAW system.

solution returned (9). The figure also illustrates the different open source software technologies used in the prototype. Text retrieval is supported by the Apache-Lucene engine. XPath and XQuery mechanisms of XML-based search are supported by the eXist native XML database framework. Basic graph isomorphism algorithms have been prototyped using the open-source Java Data Structures Library (JDSL). Similarity-based retrieval is supported by the SFW algorithm. Case composition is supported by the SHOP algorithm. These modules are implemented as LISP and JAVA servlets. The verifier and adaptation modules are currently under development. In the following subsections, we discuss our approaches for similarity-based case retrieval and planning-based case composition.

4.1. Case retrieval using similarity flooding for workflow algorithm

Similarity-based case retrieval in CODAW is based on the similarity flooding algorithm, and is called Similarity Flooding for Workflow (SFW). Similarity-based case retrieval may be initiated by a workflow designer (during manual execution) or by the process control module during automated execution of the design cycle. The input provided for case retrieval is a labeled query graph defining the task and control nodes. The output of retrieval is a collection of cases, with similar process graph structures. The similarity flooding algorithm is applicable for matching directed labeled graphs and we have adapted the same for matching workflow process models. The basic SF algorithm matches nodes in two process models, the query graph (QG) and a source graph (SG). The algorithm uses an iterative fix-point computation, results of which suggest, which node (or arc) in QG is equivalent to a node (or arc) in SG. For computing similarities, the algorithm relies on the idea that elements of two distinct graphs are similar, when their adjacent elements are similar. Similarity values are initially assigned to the elements that we may expect to match such as nodes or edges. The algorithm propagates the similarity from a node to its respective neighbors based on the topology in the two graphs. The spreading of similarities is akin to IP packets flooding a network, hence the name similarity flooding algorithm. The algorithm produces a mapping, M, which for each node in the query graph, suggests the best candidate matches in the source graph. This mapping is filtered based on a variety of domain-dependent filters to identify the best possible match between the query and source graphs.

The SFW algorithm for case retrieval is based on matching the nodes of the process graph element (as the source graph SG) of each prototypical case to the nodes of the query graph QG provided as input. In a prototypical case, the process graph element is represented explicitly as graph structure, $\mathscr{P}(N, A)$, with labeled control nodes, N_c, task nodes, N_t, concurrency nodes, N_r along with labeled arcs, A, such that $N = N_t \cup N_c \cup N_r$. This process graph structure is represented by the metamodel presented in [3] (see Appendix B for example). Our choice of the SF algorithm was guided by its ability: (a) to cope with edge and node labeling mismatches between the query and source graphs, and (b) to cope with missing graph elements in query and source graphs (unlike exact graph isomorphism matching techniques). Similarity metrics may be defined in a flexible manner to consider a variety of domain-dependent features. Further, such an inexact matching technique may be useful in retrieving process graphs from buildtime databases of commercial workflow systems, which exhibit a variety of noisy features such as the above. Exact matching techniques required costly pre-processing to obtain effective results in our preliminary studies.

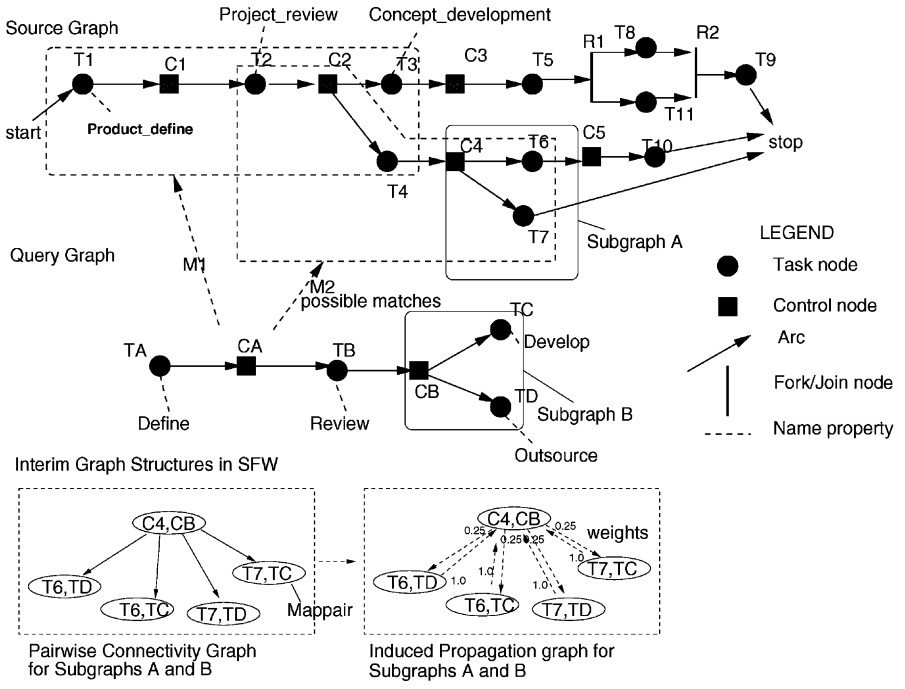

Fig. 7. Example of similarity matching.

We illustrate the SFW algorithm with an example shown in Fig. 7. The source graph (*SG*) is a variant of the product development process shown in Fig. 3. Task nodes are (labeled `T1`, `T2`, ...), decision nodes are (labeled `C1`, `C2`, ...) and concurrent nodes are (labeled `R1`, `R2`, ...). We use the labels for illustrative purposes. In the CODAW implementation, appropriate labels extracted from the `WSPGModel` tag of the case representation are used for the source graph. Note that after every task completion, a task selection step is executed by the workflow engine, indicated by control nodes. Similarly the query graph (*QG*) is labeled. Associated with each task node in either graph is a `Name` property that denotes its functionality. For example, the `Name` property of `TB` in *QG* is `Review`. Similarly, the `Name` property of each control node reflects its type, such as `AND`, `OR`, `SPLITOR`, `JOIN` and `FORK`. These are also extracted from the `WSPGModel` tag. For illustrative purposes, assume that name properties of nodes in each of the two sets `(C1, C2, CA)` and `(C2, C4, CB)` are same in the example. The first and second columns in Table 1 show a partial list of labels and the name properties for task nodes in *QG* and *SG* respectively. The third column in Table 1 is the initial similarity estimated for the labels in the example. The initial similarity between labels of the various nodes in the two graphs is estimated automatically by a rule-based name matching routine based on Natural Language Processing and string matching techniques [21]. For instance, `Define` and `Product_define` share common substrings with similar semantics and are assigned a similarity of 1.0. Such algorithms for parsing and matching are well-developed in the field of text retrieval and a discussion is beyond the scope of this paper. The fourth column in Table 1 is the similarity computed at the convergence of the iterative SFW algorithm.

Table 1
Similarity table for the example

Element of query graph	Element of source graph	Initial similarity	Final similarity
Define (TA)	Product_define (T1)	1.0	0.78
Review (TB)	Project_review (T2)	1.0	1.0
Outsource (TD)	Concept_sourcing (T4)	0.4	0.38
Develop(TC)	Concept_development (T3)	1.0	0.85
Define (TA)	Manfg_concept (T8)	0.0	0.09
Define (TA)	Project_review (T2)	0.4	0.32
Review (TB)	Concept_sourcing (T4)	0.6	0.54
Develop(TC)	Select_supplier (T6)	0.0	0.05
Outsource (TD)	Alt_strategy(T7)	0.4	0.38

From a cursory glance at the two graphs, it is possible to say that *QG* is embedded in *SG*, with two possible structural embeddings, shown as M1 and M2 in the figure. However, if one were to match the value of named properties between the two graphs, M1 is the preferred match, because the task labels are more "similar", (denoted by $\backsim$) between the two graphs. Define $\backsim$ Product_define, Review $\backsim$ Project_review and Develop $\backsim$ Concept_development, whereas the similarity in the matching M2 is much lower as the name properties are not semantically equivalent. The SFW algorithm matches elements of the two graphs based on the similarity between their graph elements and returns multiple mappings. Given a mapping threshold consisting of the number of nodes (from *QG*) matched with high-similarity, the first mapping is selected and returned as a viable match for the query. The inputs to the SFW algorithm are *SG*, *QG* and a similarity table, *T* for elements (equivalent to the third column in Table 1) of the respective graphs. The steps of SFW algorithm are summarized below (readers are referred to [30] for complete details).

Similarity Flooding for Workflow Algorithm

- InitialMap = CreateInitMap(QG, SG, T). The InitialMap is based on deriving an auxiliary data structure called an induced similarity propagation graph from *QG* and *SG*. The first step is to develop a pairwise connectivity graph (*PCG*) such that: $((x,y),p,(x',y')) \in PCG(QG,SG) \iff (x,p,x') \in QG$ and $(y,p,y') \in SG$. x,x' are nodes in *QG* connected by a directed arc, p. y,y' are nodes in *SG*, connected by a an arc, p, having the same label. Thus each node in *PCG* is an element of $QG \times SG$, called a map pair. The *PCG* for two sub-graphs, A from *SG* and B from *QG* is shown in Fig. 7. The triple C4 $\rightarrow$ T6 matches CB $\rightarrow$ TD. In the context of workflow process graphs, all arcs implicitly share the same label, namely, ControlFlow (not shown in Fig. 7 for clarity). The nodes, C4,CB form a map pair as shown in Fig. 7. Two map pairs may be connected by an edge only when the corresponding edges in each individual graph overlap. The *PCG* for just the task nodes in the example will have a subset of the 44 entries (11×4). The initial similarity between members of a map pair is provided by *T*, shown by the third column in Table 1. Decision and concurrency node similarities are binary. If their named property matches, similarity = 1 or 0 otherwise. Similarity between task nodes and control is assigned 0. This PCG is then transformed into an induced propagation graph

with the addition of two weighted (weight denoted w_i) propagation edges between each map pair (shown by dashed arrows in the induced propagation graph) that allows similarity to flow between neighboring map pair nodes. The weights $(0 \leqslant w_i \leqslant 1)$ on the edges of the induced propagation graph indicate how much of the similarity of a given pair of nodes propagates to its neighboring map pairs. For each map pair node in the induced propagation graph, with N_{total} outgoing edges, $\sum_{1<i<N_{\text{total}}} w_i = 1.0$. Initial weight, w_i, assigned to each outgoing arc $= 1.0/N_{\text{total}}$. Hence each outgoing arc from map pair, `(C4, CB)` in Fig. 7, is assigned a weight of 0.25. Incoming weights depend on the adjacent map pair nodes. Once weights are assigned, the InitialMap data structure is complete. The induced propagation graph for the example is illustrated in Fig. 7.

- `product = SFW(QG, SG, InitialMap)` executes an iterative fix-point computation as follows: Let $\sigma(x, y) \geqslant 0$ be the similarity, called a mapping, between nodes $x \in QG$ and $y \in SG$ defined as a function over $QG \times SG$. The algorithm computes σ-values iteratively. Let σ^i denote the mapping between QG and SG after the ith iteration. σ^0 denotes the initial similarity provided, shown in the third column of Table 1. The basic update formula is

$$\sigma^{i+1}(x,y) = \sigma^i(x,y) + \sum_{(a_u,p,x)\in QG,(b_u,p,y)\in SG} \sigma^i(a_u,b_u).w((a_u,b_u),(x,y)) + \sum_{(a_v,p,x)\in QG,(b_v,p,y)\in SG} \sigma^i(a_v,b_v).w((a_v,b_v),(x,y)) \quad \text{for } (a_u,b_u),(a_v,b_v) \in PCG.$$

  The iteration terminates when $\Delta(\sigma^n, \sigma^{n-1}) \leqslant \epsilon$ for $n \geqslant 0, \epsilon \geqslant 0$. A variety of variations of this update formula are discussed in [30] which we have implemented and experimented with. The resultant product is a ranked list of map pairs—a multimapping, from which many possible distinct subsets can be chosen. The fourth column of Table 1 indicates the final similarity upon convergence for the example.
- `result = Filter(product)` uses a set of similarly thresholds rules to select the best candidate match. These filtering rules are determined experimentally. Changing the threshold value affects the number of matches considered for post-processing. Specific domain knowledge may also be used to aid the filtering step. In the example, the best match(see the relative similarity values between the different choices) is `TA` $\backsim$ `T1`, `TB` $\backsim$ `T2`, `TC` $\backsim$ `T3`, `TD` $\backsim$ `T4`. SFW also produces alternative mappings such as `TD` $\backsim$ `T3` and `TC` $\backsim$ `T4`, which however have a low similarity value.

The SFW algorithm matches a pair of directed labeled graphs. When the initial similarities between the labels are all unity, the algorithm is equivalent to a graph isomorphism procedure. To search the repository for all possible matches for a query graph, we currently perform an SFW algorithm based match routine for the set of cases selected based on the name property, task and domain hierarchy. The SFW algorithm reaches a fix-point because of the directionality of arcs guiding the flow of similarity between map pair nodes in the induced propagation graph. The similarity flooding algorithm does not converge and performs poorly on *undirected* labeled graphs. The similarity-based retrieval only accounts for semantic similarity reflected by the node and edge labels and the similarity in topology of the query and source graphs. Extensions of the SFW algorithm to match collections of graphs efficiently, improve the similarity metric estimation procedure and filtering criteria are under development.

4.2. Case composition with SHOP-based planning

Case composition may be triggered (either manually or by the process control module), when retrieved cases need to be composed or a new process model needs to be generated via composition of primitive tasks. Inputs for composition include the initial and final states of a business problem defined in a planning language (defined below). The output is a set of declarative process models. Workflow case composition is based on the idea that a workflow—a sequence of tasks or a plan—defines a path in a state-space implicitly defined by the pre-conditions and post-conditions of all the tasks in the domain. The initial and final states of such a path are defined by the business goal requirements. Composition of tasks to develop a workflow is equivalent to identifying tasks in a sequence that can transform an initial world state into a final required world state. The SHOP planning algorithm relies on a domain theory consisting of composite tasks called methods and primitive tasks called operators for generating plans. A method specifies how to decompose a higher-level task into a set of subtasks. Each method is associated with various constraints that limit the applicability of the method to certain conditions on the current state and define the relations between the subtasks of the method. A method is an expression of the form $M = (h, P, TL)$, where h (the method's head) is a composite task, P is a set of preconditions, and TL is the set of M's subtasks. M is applicable to a task, relative to a current state, S defined by a set of propositions, *iff* matches (h, t, S) (i.e., h and t have the same predicates and arity and a consistent set of bindings)—there exist consistent values for variables both in S and P—such that P is satisfied. An operator is an expression of the form $O = (h, C, aL, dL)$, where h is the head, C is the set of preconditions, and aL and dL are add and delete lists. These lists define how the operators application transforms the current state S; every element of aL is added to S and every element of dL is removed from S. An operator O is applicable to a task t, relatively to state S, *iff* matches (h, t, S).

The afore-mentioned domain model of planning is supported by the declarative representation of prototypical cases, instance-level cases and primitive tasks (defined in Section 3). Each method in SHOP is equivalent to a declarative definition of a prototypical case in Section 3. Instance-level cases are grounded instances of methods (as defined above) that provide a one-level decomposition into a sequence of operators that transform a particular ground state into a final ground state. Each instance-level case C, is denoted by $C = (h, P, ST)$, where h is the head, P is the set of preconditions such that the case can be applied to the current state, ST is the actual sequence of tasks that transform the current state and consists of primitive operators. The declarative definition of a primitive task models an operator as defined above.

Fig. 8 shows the declarative description of two cases (from the domain of NPD) extracted from their corresponding XML case representations. Detailed description of the syntax is in [15]. Each case is described by the predicates that define its pre-conditions and post-conditions. One case supports the detailed design task, where a configuration of components is selected from a predefined component library, and the other case from procurement, defines a process to source particular components from a given supplier. Note that the pre-conditions apply different predicates (such as `valid-curr-state`, `is-assembly-finalized`) to the current state descriptor to ensure that the cases may be applicable in a given state. The post-conditions (also called effects) of the cases define the actual plan steps (primitive tasks are prefixed by `!`) that may be executed from the current state, namely, `(!Obtain *)`, `(!Check_new_assembly *)`,

```
;;A DESIGN CASE
 (:method (detaileddesign_case)
;;PRECONDITIONS
  ((is-valid-case CASE25)
  (currproductstate ?y)
  (valid-curr-state ?y '(C23 C27))
  (goal (goalproductstate ?g))
  (assign ?r_components (find_remaining_comps `?y `?g))
  (valid-remaining ?r_components '(C1 C2 C45 C49))
  (assign ?currdesign (check_design `?ws `?r_components)))
;;POSTCONDITIONS
 (:ordered
 (!Obtain ?r_components)
 (!Check_new_assembly ?r_components ?y)
 (!Test_assembly ?r_components ?y)
 (!Review_assembly ?r_components ?y)
 (!Finalize_assembly ?r_components ?y))
;;A PURCHASING CASE
 (:method (procurement_case)
;;PRECONDITIONS
  ((is-valid-case CASE35)
  (currproductstate ?y)
  (is-assembly-finalized ?y)
  (goal (goalproductstate ?g))
  (assign ?components (find_comps_to_purchase `?y))
  (valid-components ?components '(C1 C2 C45 C49 C23 C27))
  (assign_order_size (get_size ?y)))
;;POSTCONDITIONS
 (:ordered
   (!Place_order ?components SUPPLIER35)
   (!Receive_invoice SUPPLIER35)
   (!Receive_shipped_order SUPPLIER35)
   (!Check_components '(C2 C49))
   (!Init_pmt SUPPLIER35)
```

Fig. 8. Two new product development related cases.

`(!Test_assembly *)`, `(!Review_assembly *)` for the product design case in the figure. Note that `*` in each predicate refers to grounded values initialized in the pre-conditions.

A planning problem for SHOP is defined by an initial task network (T, S, D), where T is a set of tasks, S is the initial state and D is a domain theory consisting of operators (O), methods (M) and cases (C). A plan is the collection of primitive operators obtained by decomposing, (T, S, D). The SHOP planning algorithm performs recursive search of the planning state space via task decomposition and constraint satisfaction. The basic SHOP algorithm for solving the case composition problem, (S, T, D), where SHOP is refining a tasklist T' relative to state S and domain theory D is based on ordered task decomposition. Initially the plan, p, is empty; SHOP proceeds as follows:

The SHOP Algorithm

- **Step 1** Based on current state S, search case base for a case C that may be applicable. If so, apply case C and update S, else do Step 2.
- **Step 2** If t is primitive and has an applicable operator O, then O is applied to t, S is updated accordingly, t is removed from T' and added to the end of p. Go to Step 1. If Steps 1 and 2 are not applicable, go to Step 3.

- **Step 3** Else if t is composite and has an applicable method M, then apply M and replace t in T' with the appropriate subtasks.
- **Step 4** Else if T' is empty, then backtrack.
- **Step 5** Else fail.

The algorithm terminates when T' is empty in which case p is the solution or when SHOP tries to backtrack on a composite task t whose applicable methods have been exhausted.

Case matching is an integral step before every task decomposition phase in the algorithm. For a single state, multiple cases may match because their preconditions may be fulfilled in the current state. Choices among these cases may be based on the distance from the required goal state, estimated by comparing the add and delete lists of the operators with the goal state. In our current implementation, an applicable case is chosen non-deterministically and upon failure during search on that path, SHOP will backtrack to explore alternative branches. Methods and cases are matched based on unification of the predicates in the ground state with their respective preconditions. Planning-based composition uses all methods (prototypical cases), cases (instance-level cases) and operators (primitive tasks) to generate plans. Planning is a combinatorially difficult problem [38]. The effectiveness of the SHOP algorithms may be considerably improved with appropriate design of the predicates, operators and methods. In the worst case, the planner will explore the complete search space incurring exponential costs. In the context of CODAW, time-out mechanisms are implemented in SHOP to ensure termination and return control to the CODAW process controller. Case selection may also be performed interactively to guide the planner [32]. Cases are retrieved and presented to the user, who may select the case to apply to the current state. The state is appropriately updated and the cycle is repeated.

We have adapted a LISP implementation of the SHOP planner for case composition in CODAW. To illustrate case composition using cases shown in Fig. 8, consider a design scenario where a new product development process requires a detailed design task that combines an online catalog-based design approach with sourcing of components from a supplier. This new problem is defined by providing an initial state and a final goal state in terms of appropriate predicates. The initial state is defined by ground predicates (a partial list) shown below:

```
(GOAL (CONFIG_PRODUCT "TRANSFORM ROTARYPOWER INTO ELECTRICAL STORAGE")
  (FINALPRODUCTSTATE (HASCOMPS '(C49)))
  (ASSEMBLY_FINALIZED)
  (SUPPLIER_SELECTED)
  (ORDER_PLACED))
(CURRPRODUCTSTATE '( ))
(SUPPLIER_CATALOG S32 '(C1 C23 C24...))
(SUPPLIER_CATALOG S17 '(C19 C11 C37...))
```

The `(GOAL *)` predicate defines the function of the product to be designed and indicates that a particular component needs to be part of the final design (`(HASCOMPS *)`, the assembly is finalized, the supplier is selected and order has been placed. The `(CURRPRODUCTSTATE nil)` indicates that the process is yet to begin. Additionally the state description includes suppliers with their available components and description of component libraries (not shown). Applying the

SHOP planning methodology, results in one of the alternative plans including the combination CASE35 → CASE25 (from Fig. 8), wherein the right arrow indicates sequence. The process model on composing these two cases is (operator arguments are not shown): !Obtain → !Check_new_assembly → !Test_assembly → !Review_assembly → !Finalize_assembly → !Place_order → !Receive_invoice → !Receive_shipped_order → !Check_components → !Init_pmt. This sequential plan is then post-processed to identify concurrent tasks, such as !Check_components and !Init_pmt, by analyzing data dependencies as outlined in [7]. Alternative plans may be generated based on multiple paths through the state space. The workflow thus generated is in a declarative form and converted into an appropriate procedural process graph by instantiating the tasks with their procedural descriptions (based on the TaskImpl tag in the case representation).

5. Discussion and concluding remarks

Development of effective case retrieval and case composition algorithms is essential for the utilization of CODAW for workflow modeling in real world scenarios. The proposed case representation supports similarity-based case retrieval using the process graph formalism and planning-based case composition is facilitated using the declarative representation. Our experience in developing the case repository in CODAW has highlighted the complexity of obtaining workflow cases that may be reusable in an automated manner. Commercial workflow representations are proprietary and their buildtime XML-based representations (if available) are cumbersome to manipulate. Further, workflows created using such tools may use non-standard ontologies for labeling the workflow process models. This has motivated our development of the SFW algorithm to cope with the ontological mismatch problem. The underlying SF algorithm has proven to be reliable in large-scale conventional database schema matching problems. We have conducted extensive simulated experiments to evaluate the robustness of the SFW algorithm to changes in similarity and graph topologies. SFW is sensitive to relative values of initial similarity estimated by the string-matching algorithm. We are currently developing reliable means to assign initial similarities based on other properties (of cases) such as the different structural elements in the case representation such as the task types, inputs, outputs and history.

The development of the declarative representations, including the associated predicates and state descriptors, to support case composition is a complex task. A process graph does not provide explicit insight into the underlying design intent of why a particular task was chosen and instantiated in a given schema. It does not support reasoning about the interaction between the tasks. In contrast, a plan is a process model with state, developed from explicit encoding of the domain knowledge, which facilitates reasoning about choices of tasks and their inter-relationships. Developing appropriate declarative representations requires background domain knowledge and acquiring such knowledge is difficult. Development of such declarative representations is essential for the convergence of the Semantic web and workflow technologies in the short term. We note that our choice of first order predicate logic representation for case composition has been guided by its use in real-world planning systems and formal models of Semantic Web ontologies. Currently in CODAW, the SHOP algorithm generates a sequential workflow, which is then post-processed into a workflow with concurrent tasks by analyzing data dependency constraints. This

staged approach has been adopted to identify if a solution to a new problem exists, in a timely manner. Consideration of concurrency explicitly during the planning process increases the size of the search space and requires models for temporal and resource reasoning. We are currently experimenting with partial-order planners, that consider concurrency, to compose cases [38]. Our research on planning-based process composition techniques has been successfully used for dynamic web services composition [26]. The declarative and procedural instance-level case representations may be used to infer generic process models using learning techniques such as Inductive Logic programming and statistical machine learning [38].

This paper has described a case-based reasoning approach to support workflow modeling and design. The innovative features proposed in the paper are: (1) a case representation for workflow schemas and instances that combines both declarative and procedural representations, (2) a similarity based retrieval algorithm for retrieving process graphs of workflow schemas based on graph-based queries, and (3) the use of a domain independent AI planning technique to facilitate composition of cases into a workflow.

Our future work will continue the development and testing of the CODAW framework. We intend to further develop the adaptation and verification modules for CODAW, provide techniques for retrieval based on events, and integrate the different phases of the CBR design cycle. Enabling case composition using partial-order planners for developing workflow models with concurrent tasks is an interesting and challenging problem. Furthermore, we plan to perform real world user studies comparing the CODAW prototype with commercial workflow tools such as Oracle Workflow–11*i*.

Acknowledgements

The authors wish to sincerely thank the anonymous reviewers for their comments, which have helped to improve the quality of this paper considerably.

Appendix A. Structural elements of prototypical and instance-level cases

Elements of a prototypical case

Element, XML Tag	Description	Representation
Prototypical CaseID `WSID`	Each schema is allocated Case ID a unique identifier	A symbol (`WS2`)
Case Description `WSName`, `WSDesc`	A textual annotation or summary of the case	Free text
Instances list `WSInstances`	List of instance-level cases of schema	List of symbols
TaskList `TaskList`	A list of identifiers for primitive and composite tasks	(`collect-order`, `check-credit`)
Process Graph `WSPGModel`	Provides the workflow process model as graph	An attributed directed graph with task, control, join and fork nodes

Appendix A (*continued*)

Element, XML Tag	Description	Representation
Inputs `WSInputs`	List of arguments to initiate workflow	Each argument is an attributed data entity such as `Customer`
Outputs `WSOutputs`	List of arguments after completion of workflow	Each argument is an attributed data entity such as `Completed_transaction`
Preconditions `WSPreConds`	Conditions that need to exist in current organizational state to instantiate prototype case	Described as first order logic predicates `(available_customer_data ?customer)`
Postconditions `WSPostConds`	Conditions that will exist in state after completion of case	Described as first order logic predicates `(order_shipped ?customer ?date)`
Resource Types `ResTypeList`	List of resource types, roles used in the case	Attributed data objects
Preference ranking `WSRating`	Usage ranking for filtering choices during retrieval	A number

Elements of a primitive task

Element, XML Tag	Description	Representation
Task ID `TID`	Unique task identifier across all domain descriptions	A symbol `TI-21`
Task Parameters `TParams`	List of state variables that are used by task	A parameter list such as (?name ?address ?order)
Task Preconditions `TPreConds`	Conditions that need to exist in the current state to initiate task	Described as first order logic predicates `(previous_task_completed TK-15)`
Task Postconditions `TPostConds`	Conditions that will exist in state after completion of task	Described as first order logic predicates `(customer_registered ?c)`
Task Implementation `TImpl`	Refers to executable procedural code for task or manual steps	Implemented as a method call or named procedure

Elements of an instance case

Element, XML Tag	Description	Representation
Instance CaseID, `WIID`	Unique identifier for an instance	A symbol (`WI22`)
Inputs, `WIInputs`	Values for input arguments to case	Actual data element, `Customer_no 121`

(*continued on next page*)

Appendix A (*continued*)

Element, XML Tag	Description	Representation
Outputs, `WIOutputs`	Values for output arguments	Actual data element, `Transaction_ID 221`
Initial State `WIInitState`	Describes state when case was started	Described in terms of first-order predicate logic
Final State `WIFinalState`	Describes state when case was completed	Described in terms of first-order predicate logic
State History `WIHistory`	A sequence of interim states recorded at specific points	Stored in special state data-structures for predicate values
EventsList `WIEventlist`	List of systemic and application-specific events	Stored as attributed event data types
Exception-Handling Actions, `WIActions`	List of failure handling actions	Stored as attributed action data entities
Current Case Status `WIStatus`	Describes execution status (halted, in-progress etc.)	Stored as attributed state description entities
Performance Metrics `WIMetrics`	Describes metrics of case such as flowtime, resource utilization	Time in minutes, percentages
Agents `WIAgents`	Maps specific roles to agents used in the case	Attributed data entities for agents
Resources `WIResources`	Maps specific resource instances to tasks used in the case	Attributed resource instances

Appendix B. Prototypical case representation of a NPD workflow schema

```
<?xml version="1.0"?><!DOCTYPE WorkflowSchema []>
<WSCase> <WSID> WS2</WSID>
<WSName> Market-Pull Workflow </WSName>
<WSType> Product Development</WSType>
<WSDesc> A generic product development process used
in routine product design. </WSDesc>
<TaskList> ((TID-1 Project_Selection)(TID-21 Product_Definition)
(TID-31 Project_Plan_review),
(WS21 Detailed_design)..)</TaskList>
<!List of Composite tasks and associated subprocess schema>
<ComponentWorkflows>(Detailed_design   WS21)   </ComponentWorkflows
Modified>
<!Instances of this workflow>
<WSInstances> (WI1 WI22 WI23 ...) </WSInstances>
<WSInputs> (Design_problem Budget) </WSInputs>
<WSOutputs>(Design_Solution Estimated_cost Actual_Cost)</WSoutputs>
<!—Problem, Budget, Solution are attributed objects>
```

```
<WSPreConds> (available_new_problem ?designprob) (prioritized ?design-
prob)
(soln_reqd ?designprob) </WSPreConds>
<WSPostConds> (completed_workflow WS2) (available_newsoln ?newsoln)
  </WSPostConds> <WSRating> (times_used 12) (average_time 7mos)...
  </WSRating>
<WSPGModel>    <NodeList>
<Tasknodes> (TID-1, TID-21, TID-31...) </Tasknodes>
<Decisionnodes> (DN-1 SEQ),(DN-2 SEQ)..</Decisionnodes>
<Concurrencynodes> (FK-1 FORK) (JN-1 JOIN) </Concurrencynodes>
</Node-List>
<EdgeList> (START TID-1) (TID-1 DN-1) (DN-1 TID21)..</EdgeList>
</WSPGModel>
<Tasks> <Task> <!—For each task its declarative and procedural defn is
provided>
<TaskType> Business </TaskType>
<TaskName> Project_Selection</TaskName>
<TaskDesc> Selects a list of new product ideas to work on </TaskDesc>
<TaskID> TID-1 </TaskID> <!—Task descriptions—>
<TParams> <Param> ?project_list </Param> <Param> ?total_budget
</Param>
<Param> ?resource_list</Param> </TParams>
<TPreConds> <!—Each predicate is FOL predicate with appropriate variables>
<Predicate> (available ?project_list) </Predicate>
<Predicate> (available ?resource_list) </Predicate>
</TPreConds> <TPostConds>
<Effect> (add (new_proj_list ?new_list)) </Effect>
<Effect> (add (new_budget ?new_budget)) </Effect>
</TPostConds> <TImpl>
<Agent> General_Manager </Agent>
<Agent> Marketing </Agent>
<Agent> Engg_Design </Agent>
<Agent> Manfg </Agent>
<Agent> QA </Agent>
<Agent>Purchasing</Agent>
<Agent> Customer_Service</Agent>
<Procedure> <ProcedureName> Select_Project </ProcedureName>
<ProcedureSource> HandBook </ProcedureSource>
<Implementation_type> Manual_Team_Execution </Implementation_type>
</Procedure> <Inputs>
<DataItem> budget </DataItem>
<DataItem> resources </DataItem>
<DataItem> projects </DataItem>
</Inputs> <Outputs>
```

```
<DataItem> selected_projects </DataItem>
<DataItem> remaining_budget </DataItem>
</Outputs> </TImpl> </Task>....
<WSResTypeList> (Engineers 10) (Draftsmen 7)...</WSResTypeList>
</WSCase>
```

Appendix C. Instance-level case representation

```
<?xml version="1.0"?><!DOCTYPE WorkflowInstance []>
<WIID> WI22</WIID>
<WIInputs> Honeywell_AeroStarter, 75000$ </WIInputs>
<WIOutputs> AeroStarter_PartNumber_1729, 60000$, 85000$ </WIOutputs>
<WIInitState> (allocated_problem HAstarter) (allocated_budget 75000)
</WIInitState>
<WIFinalState> (newdesign PN1729) (cost 85000) </WIFinalState>
<WIHistory> </StateList>
<State> S2 (date jan/25/03) (Project_initiated)(team_selection_
begun)</State>
<State> S3 (date jan/29/03)(team_size 10)(team_selection_complete)
(request_new_employee enggr) ...
</State></StateList>
<WIEventsList>
<!Events classifed by type, date occurred, desc, which task, possible
cause, repair action>
<Event> DB_down, feb/14/03, "Project database server down", TID-31,
  "corrupt_file", Recover_action_21</Event>
<Event> Agent_unavailable, march/19/03, "Enggr. resigned", TID-42,
  "corrupt_file", Recover_action_45</Event>
.....
</WIEventList>
<!—Action descriptions include type of action, date initiated, status>
<WIActions>
<Action> Recover_action_21, perform_backup_recoveryof element,
  initiated feb/17/03, completed feb/21/03 </Action>
<Action> Recover_action_45, initiate_new_hire_process,
  initiated april/11/03, inprogress </Action>
</WIActions> </WIHistory>
<WIStatus> Complete, 100% </WIStatus>
<WIMetrics> 25% Schedule overrun, 15% Overrun budget, Design_accepted
</WIMetrics>
<WIAgents> (Agent EmpNo_12345003 General_Manager) .... </WIAgents>
<WIResources> (Engineers_used 17) (Overhead_consumables 25000)..
</WIResources>
```

References

[1] W.M.P. van der Aalst, K. van Hee, Workflow Management, MIT Press, 2002.
[2] W.M.P van der Aalst, A.P. Barros, A.H.M. ter Hofstede, B. Kiepuszewski, Workflow patterns, Technical Report FIT-TR-2002-02, Queensland University of Technology, Brisbane, Australia, 2002. Available from: <http://www,tm.tue.nl/it/research/patterns>.
[3] W. van der Aalst, A. Kumar, XML-based schema definition for support of interorganizational workflow, ISR 14 (1) (2003) 23–46.
[4] W. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. Weijters, Workflow mining: a survey of issues and approaches, Data & Knowledge Engineering 47 (2003) 237–267.
[5] A. Aamodt, E. Plazas, Case-based reasoning: Foundational issues, methodolgical variations, and system approaches, AI Communications 7 (1) (1994) 39–52.
[6] P.A. Bernstein, A. Halevy, R. Pottinger, A vision for management of complex models, ACM SIGMOD Record 29 (4) (2000) 55–68.
[7] C. Backstrom, Finding least constrained plans and optimal parallel executions is harder than we thought, in: C. Backstorm, E. Sandewall (Eds.), Proceedings of the European Workshop on Planning—EWSP'93, 1993.
[8] A. Basu, A. Kumar, Research commentary: workflow management issues in e-business, Information Systems Research 13 (1) (2002) 1–14.
[9] H. Bergmann, R. Munoz-Avila, M. Veloso, E. Melis, Case-based reasoning applied to planning tasks, in: B.M. Lenz, Bartsch-Spörl, H. Burkhard, S. Wess (Eds.), Case-Based Reasoning Technology from Foundations to Applications, Springer, 1998.
[10] J. Blythe, E. Deelman, Y. Gil, Planning for workflow construction and maintenance on the grid, in: Proceedings of ICAPS'03 Workshop on Planning for Web Services, Trento, Italy, 2003.
[11] C. Bussler, A. Maedche, D. Fensel, A conceptual architecture for semantic enabled web services, ACM SIGMOD 31 (4) (2002).
[12] F. Casati, S. Castano, M. Fugini, I. Mirbel, B. Pernici, Using patterns to design rules in workflows, IEEE Transactions in Software Engineering 26 (8) (2000).
[13] F. Casati, M. Shan, Dynamic and adaptive composition of e-services, Information Systems 6 (3) (2001).
[14] L. Fahey, R. Srivastava, J.S. Sharon, D.E. Smith, Linking e-business and operating processes: the role of knowledge management, IBM Systems Journal 40 (4) (2001) 889–906.
[15] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL—the planning domain definition language, 1998. Available from: <http://www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator1.html>.
[16] M. Gruninger, C. Menzel, The process specification language (PSL): theory and applications, AI Magazine (2003) 63–74.
[17] S. Henninger, K. Baumgarten, A case-based approach to tailoring software processes, in: Proceedings of ICCBR 2001, vol. 2080, LNAI, 2001, pp. 249–262.
[18] J. Herbst, D. Karagiannis, Integrating machine learning and workflow management to support acquisition and adaptation of workflow models, in: Ninth International Workshop on Database and Expert Systems Applications, 1998, pp. 745–752.
[19] IDS Scheer AG, ARIS Design Platform, 2003. Available from: <http://www.ids-scheer.com>.
[20] G. Joeris, O. Herzog, Managing evolving workflow specifications. in: Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, 1998, pp 310–319.
[21] D. Jurafsky, J.H. Martin, Speech and Natural Language Processing, Prentice-Hall, 2000.
[22] J.L. Kolodner, Case-Based Reasoning, Morgan Kaufmann Publishers, San Mateo, CA, 1993.
[23] D.B. Leake (Ed.), Case-Based Reasoning: Experiences, Lessons, and Future Directions, The AAAI Press/The MIT Press, 1996.
[24] K. Lenz, A. Oberweis, Modeling interorganizational workflows with xml nets, in: Proceedings of the 34th Hawaii International Conference on System Sciences, vol. 7, 2001.

[25] T. Linden, Representing software designs as partially developed plans, in: M. Lowry, R. McCartney (Eds.), Automating Software Design, AAAI Press/The MIT Press, 1991, pp. 603–625.
[26] T. Madhusudan, N. Uttamsingh, A declarative approach for composition of web services in dynamic environments, Decision Support Systems, under review.
[27] M.L. Maher, A.G. De Silva Garza, Case-based reasoning in design, IEEE Expert 12 (2) (1997) 34–41, Special issue on AI in Design.
[28] T. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. Osborn, A. Bernstein, G. Herman, M. Klein, E. O'Donnell, Tools for inventing organizations: towards a handbook of organizational processes, Management Science 45 (3) (1999) 425–433.
[29] K. Mannel, From UML to BPEL, http://www-106.ibm.com/developerworks/webservices/library/ws-uml2bpel/, 2003.
[30] S. Melnik, H. Garcia-Molina, E. Rahm, Similarity flooding: a versatile graph matching algorithm and its application to schema matching, in: Proceedings of the 18th International Conference on Data Engineering, 2002.
[31] S. Mukkamalla, H. Munoz-Avila, Case acquisition in a project planning environment, in: Proceedings of 6th European Conference on CBR, LNAI, vol. 2416, Springer, 2002.
[32] H. Munoz-Avila, D. Aha, D. Nau, R. Weber, L. Breslow, F. Yaman, SiN: integrating case-based reasoning with task-decomposition, in: Proceedings of International Joint Conference on AI, AAAI, Seattle, WA, USA, 2001.
[33] H. Munoz-Avila, F. Weberskirch, Planning for manufacturing workpieces by storing, indexing and replaying planning decisions, in: Proceedings of the 3rd International Conference on AI Planning Systems (AIPS-96), 1996.
[34] D.S. Nau, Y. Cao, A. Lotem, H. Munoz-Avilla, SHOP: simple hierarchical ordered planner, in: Proceedings of IJCAI-99, 1999, pp. 968–973.
[35] PROMATIS Software GmbH, INCOME Suite, 2003. Available from: <http://www.promatis.com>.
[36] H. Reijers, Design and Control of Workflow Processes, in: LNCS, vol. 2617, Springer-Verlag, 2003.
[37] U. Reimer, A. Margelisch, M. Staudt, Eule: a knowledge-based system to support business processes, Knowledge-based Systems 13 (5) (2000) 261–269.
[38] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, second ed., Prentice-Hall, 2003.
[39] A.W. Scheer, ARIS—Business Process Modeling, Springer-Verlag, 1999.
[40] E.A. Stohr, J. Leon Zhao, Workflow automation: overview and research issues, Information System Frontiers 3 (3) (2001) 281–286.
[41] G. Valiente, Algorithms on Trees and Graphs, Springer-Verlag, Berlin, 2002.
[42] M. Voorhoeve, W. van der Aalst, Ad-hoc workflow: problems and solutions, in: Proceedings of the Eighth International Workshop on Database and Expert Systems Applications, 1997, pp. 36–40.
[43] C. Wargitsch, Workbrain: merging organizational memory and workflow management systems, in: Workshop on Knowledge-Based Systems for Knowledge Management in Enterprises, 1997.
[44] D.S. Weld, Recent advances in AI planning, AI Magazine (1999) 55–68.

Therani Madhusudan is an Assistant Professor at the MIS Department, University of Arizona. He holds Ph.D. (1998) and M.S. degrees (1994) from Carnegie-Mellon University and a B.Tech. (1990) from the Indian Institute of Technology, Madras, India. Prior to joining the University of Arizona, he was a lead systems architect for Engineering Knowledge Management at Honeywell International, South Bend, IN. His research focuses on the development of knowledge-based tools to support the design and management of complex hardware and software systems. He has published over 20 refereed research articles in academic conferences and journals in the areas of Workflow Management, Product Lifecycle Management and Engineering design automation.

J. Leon Zhao is Associate Professor and Honeywell Fellow of MIS, University of Arizona. He holds Ph.D. degree from University of California, Berkeley, M.S. degree from University of California, Davis, and Bachelor degree from Beijing Institute of Agricultural Mechanization. He has previously taught in Hong Kong University of Science and Technology and College of William and Mary. He has published over 80 referred research articles in academic conferences and journals including Management Science, Information Systems Research, Communications of the ACM, and Journal of Management Information Systems. He is associate editor for Electronic Commerce Research and Applications, International Journal of Web Services Research, and International Journal of Business Process Integration and Management. He also serves on the editorial board of Journal of Database Management. He is a co-chair of the Second Workshop on e-Business, 2003.

Byron Marshall is a doctoral student in AI Lab of the Management Information Systems department at the University of Arizona. He received his undergraduate degree in Business Administration-Computer Applications and Systems from California State University, Fresno 1988 and an MBA degree with emphasis in Accounting from California State University, Fresno 1995 with distinction. His research interests involve the automatic processing of node-link semantic graph representations of knowledge to support learning and knowledge acquisition processes.