PWWM: A Personal Web Workflow Methodology

Marsha Chechik, Jocelyn Simmonds, Sotirios Liaskos, Shiva Nejati, Mehrdad Sabetzadeh, and Rick Salay

Shiva and Mehrdad are with Simula Research Lab, Norway. Jocelyn is with Departamento de Informática, Universidad Técnica Federico Santa María, Chile. Sotirios is with Department of Computer Science, York University, Canada. Rick and Marsha are with Department of Computer Science, University of Toronto, Canada.

Abstract. The personal web vision promises to give users a highly personalized experience on the web. This paper proposes and describes a Personal Web Workflow Methodology, designed to elicit, operationalize and execute a personal web user's goals. Our approach relies heavily on our prior research in goal modeling and operationalization, model matching and merging, and web service monitoring and recovery. We integrate this research with the social networking concept of crowd-sourcing to create a novel methodology for allowing users to produce customized workflows in order to accomplish their unique goals.

1 Introduction and Motivation

Personal web is ultimately a way to give every user a truly personalized experience on the web. From remembering her preferences of sites and policies, maintaining her context, organizing the most essential information, to allowing collaboration and information sharing with her family and friends, the vision of ultimate personalization seems almost within reach.

In our position paper presented at the Personal Web workshop [17], we proposed the particular area of our interest in this context as *trying to elicit and execute a personal web user's goals, through preferred information collection devices and with cooperation of trusted individuals*. For example, traditional web applications such as commerce and banking offer a particular interaction with the user and his/her data. Data is stored in the database of a particular application (e.g., shopping list or wish list), and the user is being offered a particular workflow that determines the interaction of the user with the system (e.g., on amazon.com, such things include looking for something, doing a price comparison, determining a particular vendor to go with, choosing the type of shipment and the payment method).

Instead, as users, we may want to use parts of the different applications which are useful to us, and then combine them in our own, personal ways. For example, when buying electronics, a savvy Canadian consumer may want to first check amazon.com to look at the models and reviews. Amazon.ca has a much smaller product selection, and very likely will not carry the desired product. Instead, she would look for the equivalent models on other Canadian retail sites. After comparing prices and shipping options, she may want to consult her friends and/or family, and then hit the "pay" button. Wouldn't it be nice if such a process could be stored and repeated whenever the user needs to execute it? This will ensure that steps are not skipped, and our consumer gets the best deal.

Once such workflows are explicated and stored, they may become updated as additional information becomes available. For example, our shopper may hear of additional sites where reliable research can be conducted, additional sources of online coupons to check, or may want to integrate portions of personalized workflows of other Canadian consumers.

Generic workflows can be created as well, and stored on the web in a manner similar to existing customizable phone apps. Some examples of those can be a "web for a Canadian shopper" workflow, or a "Dinner and a Movie" workflow, involving choosing an interesting movie, a time that works and a location which is reasonable to get to and that has a restaurant close by that the person executing the workflow would like to visit and that has seating available in time to catch the movie, all the while coordinating with the persons's date, and restaurant/movie review sites.

In this paper, we propose and describe PWWM – a Personal Web Workflow Methodology that aims to elicit the user's goals for a particular task and create a customizable workflow to accomplish it. Our proposal builds on our areas of expertise: goal modeling and operationalization [44, 45], model matching and merging [57, 69], and web service monitoring and recovery [72]. Specifically, we show how to use and adapt techniques developed in the three areas above in order to elicit user's goals, synthesize possible workflow models, find and merge these with crowd-sourced generic workflows, use planning to produce optimal plans through these workflows, identify relevant web services which can execute various parts of the plan, create custom orchestrations of these services, monitor them dynamically against a variety of user and vendor policies and constraints, and, if a failure is discovered, perform recovery and/or produce an alternate plan. We also rely heavily on a social networking concept of "crowd-sourcing", to help fill the gaps.

We expect our collection of techniques to enable creation of interesting and truly customizable user experiences, while maintaining a degree of quality control over correctness of the execution of the proposed plans and workflows.

While our methodology centers around *customization*, it does not yet address the issue of *collaboration* – where multiple users perform steps towards achieving a common goal.

In the rest of this paper, we describe an example user problem and illustrate a possible user experience with PWWM as she attempts to solve her problem (Section 2), overview our methodology (Section 3), give the necessary background on the three enabling techniques (Section 4), describe assumptions and detail the steps in the methodology (Sections 5-9), outline some research challenges stemming from our proposal (Section 10), discuss related work (Section 11) and finally conclude in Section 12.

2 Motivating Example

To help us motivate our Personal Web vision, we describe a simple example and then show how we envision the user experience with it.

Problem. Consider the following scenario. Our (Canadian) user is six months pregnant and wants to make sure that the baby, once she arrives, will have a safe place to sleep. She is unsure about her preferred options: a bed share? a bassinet (only if she can borrow it!)? but ultimately, she needs to get her baby a crib. Quality cribs are durable but expensive, and take a while to get once ordered. So, she wants to try to buy a second-hand crib. The easiest way to get one is through a local online classified ads, such as craigslist.org, since she can go to the vendor in person and inspect it before making a decision. Our user also knows that quality cribs take 6 weeks to arrive when ordered, so she can only keep looking at used cribs for another 1.5 months. If that (soft) deadline passes, the user will have no choice but to go to a retailer that has cribs in stock and buy whatever they have – clearly not a good choice but might be the only option for meeting the hard deadline – having a crib once the baby arrives. The user can get a crib from a retailer in Canada or the US, but in the latter case should be aware of additional import charges and, most importantly, additional delivery time.

In addition, the user may have a list of preferences: (a) the user prefers a used crib but if none are available within 1.5 months, she will purchase a new one (although what if a perfect used crib becomes available within days of placing an order for a new crib. Can that order be cancelled?); (b) the user wants to avoid shipping from the US in order to avoid customs delays as well as extra taxes.

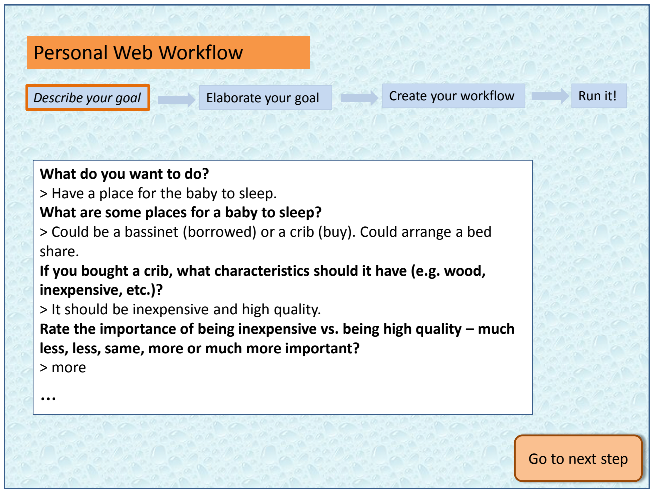
To accomplish this scenario, the user needs to interact with various services/sites:

- Research: product databases, review sites, user groups and forums.
- Purchase: auction sites, online classified ads, online retailers (and of course the related payment processing).
- Shipping: shipping estimator, shipping, truck rental.
- Utilities: currency converter, online spreadsheet, email, calendar, task lists.

User Experience. Our user aims to find a place for her future baby to sleep by using an online implementation of PWWM. Upon invoking the tool, she enters the *elicitation* step where she can describe her goal. For example, in the screen in Figure 1 she is asked a series of questions using natural language. At this point, some information about the goal is known but it is still too high level for creating an executable flow that calls real web services. For example, there is unlikely to be a service provider that can directly provide a "find a place for a baby to sleep" service, and even the sub-goal "buy a crib" may not have enough detail.

The user then moves to the *refinement* step of the methodology, where she can search the web for information that can elaborate these high-level goals to a greater level of detail. She is presented with a summary of the high level goals and can select one for further elaboration (Figure 2). For example, if she selects "buy crib", then the web is searched for different approaches for achieving this goal (Figure 3). If she selects one of these approaches, she can look at its details and configure or customize it further. This approach is then retained by PWWM and our user can elaborate another goal, and so on.

After the user has elaborated the goal model to a sufficient level of detail, a workflow can be created in the *planning* step of PWWM. The user view of this is illustrated in Figure 4. The workflow creation is automatic and takes into account all the user's



What do you want to do?
> Have a place for the baby to sleep.
What are some places for a baby to sleep?
> Could be a bassinet (borrowed) or a crib (buy). Could arrange a bed share.
If you bought a crib, what characteristics should it have (e.g. wood, inexpensive, etc.)?
> It should be inexpensive and high quality.
Rate the importance of being inexpensive vs. being high quality – much less, less, same, more or much more important?
> more
...

Fig. 1: User view of the PWWM elicitation step.

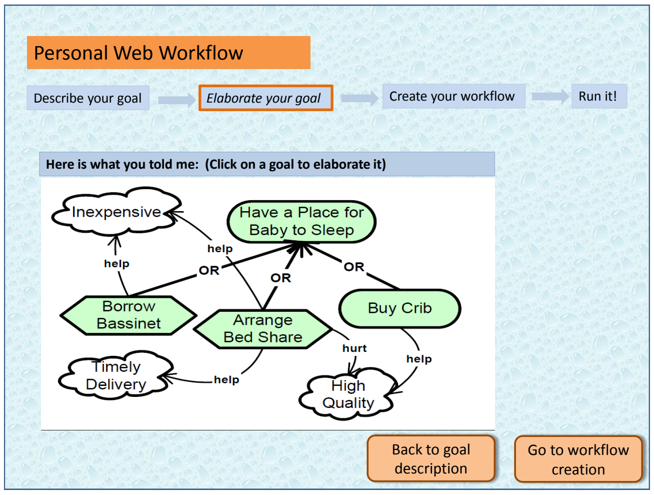
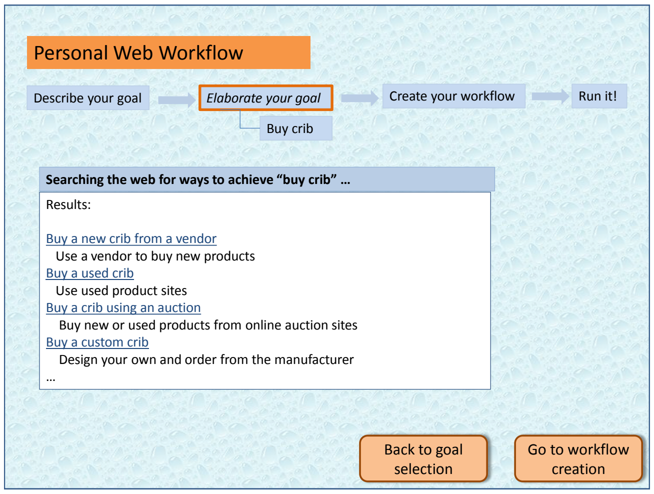


Fig. 2: User view of the PWWM refinement step.

preferences and the information she provided in the elicitation and refinement steps. Some preference information comes from a general user profile that we assume exists and contains information about the user's preferred websites, credit cards, etc. Other preference information comes from the elicitation step – for example, in Figure 1, the user stated that the crib being "inexpensive" was more important to her than being "high quality".

The user is presented with a ranked list of several workflows. The more a particular workflow satisfies her preferences, the higher is its ranking. The user chooses one (likely the top), which then gets converted into an orchestration of web services to execute it. Alternatively, she can change (i.e., relax or revise) some of her preferences, to produce

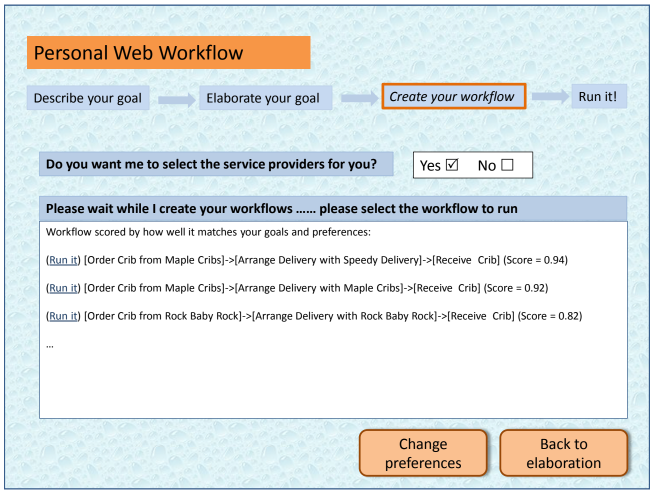


Searching the web for ways to achieve "buy crib" …

Results:

Buy a new crib from a vendor
Use a vendor to buy new products
Buy a used crib
Use used product sites
Buy a crib using an auction
Buy new or used products from online auction sites
Buy a custom crib
Design your own and order from the manufacturer
…

Fig. 3: Web search for crowd sourced goal models.



Do you want me to select the service providers for you? Yes ☑ No ☐

Please wait while I create your workflows …… please select the workflow to run

Workflow scored by how well it matches your goals and preferences:

(Run it) [Order Crib from Maple Cribs]->[Arrange Delivery with Speedy Delivery]->[Receive Crib] (Score = 0.94)

(Run it) [Order Crib from Maple Cribs]->[Arrange Delivery with Maple Cribs]->[Receive Crib] (Score = 0.92)

(Run it) [Order Crib from Rock Baby Rock]->[Arrange Delivery with Rock Baby Rock]->[Receive Crib] (Score = 0.82)

…

Fig. 4: A rendering of the PWWM planning step.

different workflows. This is accomplished by choosing the appropriate activities in the planning screen (see Figure 4).

Since each step of the workflow requires interaction with different service providers, these providers have to be selected, and the user is given the choice as to whether she wants to be involved in their selection (or whether they should be computed from user preferences and/or crowd-sourced quality ratings). If she does, then she is presented with lists of service providers as needed, dynamically, during the execution of the workflow.

Since the Web is intrinsically unreliable, our best efforts to ensure that the user-produced workflow is correct (i.e., will achieve her goal) may fail. To mitigate that, PWWM actively monitors the workflow for potential failures during execution. If a failure occurs, the framework attempts to recover for the failure either by prompting the user to select another vendor, to choose an alternative plan, to change her preferences so that different plans can be computed. Recovery often involves the use of *compensation*, e.g., cancelling a crib order with one company (and possibly incurring a restocking fee) before placing it with another.

3 Realizing Personal Workflows: Methodology

The PWWM ultimately allows the user to utilize his/her preferences of sites, vendors and policies, as well as to specify and operationalize her goals. The outcome becomes an executable orchestration of web services with a number of monitors checking whether user and vendor policies are being satisfied. In the case that user preferences change or some of the monitors fail, the system can either produce a different workflow or enable recovery.

The key challenge for this methodology is to provide a way to shield the user from the complexity of creating an executable workflow while still guaranteeing that it satisfies the user's goals. To achieve this, we adapt and integrate different research ideas and web technologies including work on configuring personal software using goal models [44,45], model merging [57,69], monitoring and recovery of web service orchestration [72], together with a social networking concept of "crowd-sourcing".

The crowd-sourced information helps the user refine her high-level goals, choose between the different refinements, determine vendor policies (including compensation), rank the vendors and find positive/negative stories capturing experiences with sets of vendors. In addition, crowd-sourcing can help the user define her personal configuration information such as favorite sites, desired policies, preferences (global or defined locally for a particular task), etc.

Figure 5 shows the personal web workflow methodology at a high level. The process begins with the Elicitation step where a high-level goal model representing the user's objectives is elicited from the user. In the Refinement step, this model is elaborated into a detailed goal model by using relevant crowd-sourced models. Then, in the Planning step, the detailed goal model is analyzed and a sequence of web tasks (i.e., a web orchestration) that satisfies the customer's goals is created. Finally, in the Execution step, this sequence of tasks is executed with the user's interaction. As the sequence runs, it is monitored against a potential violation of user or vendor policies, pre- and postconditions, availability of individual services, etc. As a failure is detected, the system attempts to recover by going back to its previous state or asking the user to choose another plan.

In the remainder of this paper, we discuss steps of the proposed methodology in more detail.

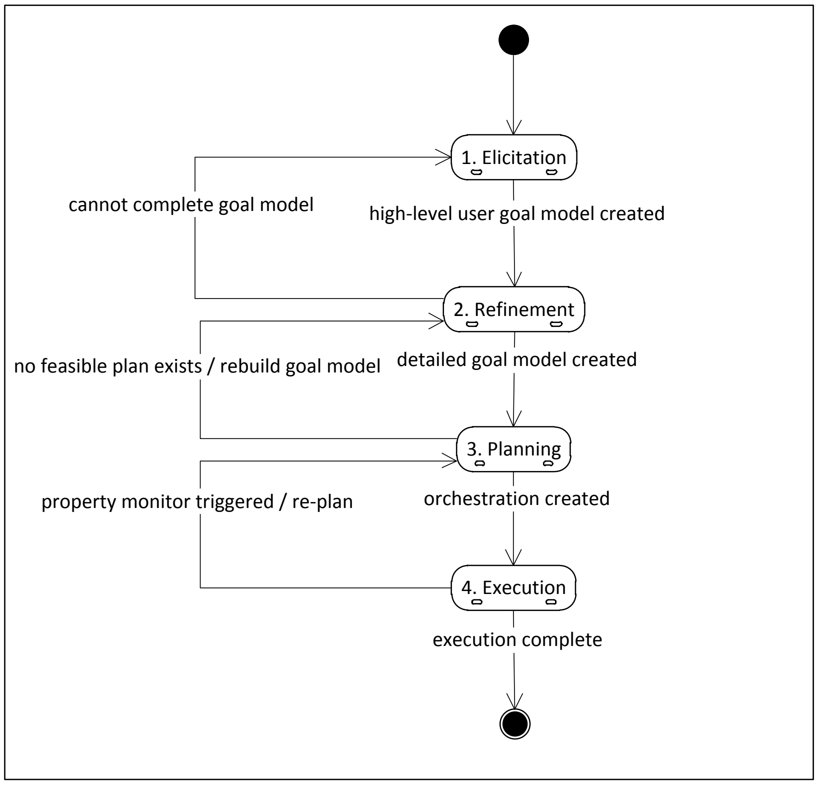

1. Elicitation
cannot complete goal model
high-level user goal model created
2. Refinement
no feasible plan exists / rebuild goal model
detailed goal model created
3. Planning
property monitor triggered / re-plan
orchestration created
4. Execution
execution complete

Fig. 5: The high-level steps of PWWM.

4 Background

In this section, we provide the necessary high-level background on the techniques used in our methodology: goal modeling and operationalization, model matching and merging, and web service monitoring and recovery.

Goal models. Goal models are the means by which user needs and preferences are captured and reasoned about. They have been found to to be effective in bridging high-level expressions of stakeholder goals with the low-level human or system activity that is required to achieve those goals [20, 55]. In *i**, the dominant goal modeling framework [88] which we adopt here, this bridging is diagrammatically represented through goal decomposition structures. Thus, high level expressions of stakeholder goals (e.g., "buy a crib") are recursively decomposed into subgoals and eventually into tasks (e.g., "provide credit card information"). Two types of decomposition can be used: AND decomposition where a goal is decomposed into a sequence of simpler goals or tasks (e.g., the goal "buy crib" can be decomposed into the sequence "select crib", "pay for crib", "ship crib") and OR decomposition where a goal is decomposed into alternative ways to achieve it (e.g., the goal "pay for crib" can have alternatives "pay through credit card", "pay through money order".). Goal models distinguish between *hard goals*, indicating a well-defined satisfaction condition (e.g., "crib must be made in Canada") and *soft goals*, indicating a desirable state without clear testing criteria (e.g., "the crib should be delivered promptly"). Positive or negative contribution links, called "help" and "hurt" links, are then drawn from different types of goals to soft-goals to show how satisfaction of the former is believed to influence satisfaction or, respectively, denial of the latter.

Goals, Alternatives and Preferences. The AND/OR decomposition structures have been shown to be remarkably useful for representing large numbers of alternative solutions by which high level goals of stakeholders can be achieved [45, 55]. These solutions come in the form of *plans*, which are sequences of leaf level tasks that satisfy the AND/OR decomposition structure and possible precedence constraints between the tasks.

A goal model may imply a great number of possible plans, but which of them are best for a given situation at hand and how can we find them? Using *preferences*, we can represent things that stakeholders desire to be true but are not mandatory, while achieving their main goal [46]. Thus, while "buy a crib" is a mandatory goal and no plan that does not satisfy it is a viable solution, desires such as "crib be hand-made" or "quick delivery" may be preferences, in that, solutions that don't satisfy them may still be accepted. Furthermore, to express the relative importance between preferences, *priority* specifications over them are possible by constructing weighted linear combinations that must be optimized. Given a user profile containing relevant preferences and priorities, powerful preference-based AI planners [75] can be employed in order to identify plans that best satisfy the profile. In this way, we are able to connect high level stakeholder attitudes and desires with descriptions of complex low-level activity that best satisfy those desires.

Model matching and merging. *Match* and *Merge* are two important model management operators with a key role in supporting the distribution and coordination of modeling activities [11,57]. The Match operator (sometimes also referred to as Map) is used to find commonalities between models. The resulting relationship is an explicit statement of how two models overlap in their content. For most types of models, it is a heuristic operator, meaning that the relationship produced by Match may miss some correct correspondences between model elements or identify some incorrect correspondences. As a result, the matching outcome typically needs to be adjusted by a user.

In our context, Match is used mainly as a prerequisite for the Merge operator, whose purpose is to unify the overlaps between a set of models and create a single holistic model. Model merging often becomes necessary when one wishes to gain a unified perspective over a set of models, to analyze the relationships between the models, or to check that models fit together in a consistent manner. In addition to the unification of overlaps, the Merge operator is often expected to satisfy several additional criteria. Some of these criteria are (1) *Completeness*: If a concept appears in one of the source models, it is represented in the merged model as well [9]; (2) *Minimality*: Merge shall not introduce new information that is not already present in or implied by the source models; and (3) *Logical Preservation*: Merge shall support the expression and preservation of logical properties. For example, for goal models, one may want to preserve the dependencies between the goals, to ensure that the intended meaning of the source models is properly captured in the merge. The Merge operator that we apply to goal models in this article (developed previously in [68]) meets all the above criteria.

Runtime monitoring and recovery. The goal of *runtime monitoring* is to check whether an application violates a given specification of its behaviour during execution. In this work, we assume that this behaviour is specified as a set of *desired* (what the system should exhibit) and *forbidden* (what the system should not exhibit) behaviours. Specifications of such behaviours come from a variety of sources: positive and negative user stories in crowd-sourced models, desired workflows defined by service providers, user preferences, pre- and postconditions of individual web services, user goal models, etc.

To create monitors, we translate these specifications into deterministic finite state machines (FSMs) and then *register* these with the execution environment. Monitors can become dynamically enabled (e.g., to monitor new properties) and disabled (e.g., to reduce the monitoring overhead or when a particular property is no longer relevant). During execution, the monitoring environment captures events as they pass between the application and its environment and uses these to update the state of the registered monitors. When any of the monitors reach their *accepting state*, this signifies that the application has executed an undesired scenario, and a violation needs to be reported. In our pictorial notation, accepting states are colored red and shaded horizontally. For example, state 3 of monitor M_1 in Figure 14 is red, indicating that the sequence $\mathsf{pay}, \mathsf{cash}$ is forbidden. Self-loop in state 1 indicates that if the system sees any event in its alphabet (denoted Σ) other than pay, it should remain in the same state.

In addition, monitors can be used to detect *desired* sequences of events, which we pictorially represent using green states shaded vertically (not all states need to have these). For example, state 4 of monitor M_1 is green, indicating that sequences $\mathsf{pay}, \mathsf{credit}$ are desirable. In order to reason about the unexpected termination of desired sequences of events, we have added a new system event TER, produced when the application terminates (regardless of the reason). For example, monitor M_7 in Figure 15 goes from state 1 to 2 on a TER event, indicating that the application terminated before the $\mathsf{receiveDelivery}$ event occurred.

Once an error is detected at runtime, our method can propose recovery plans [72]. Availability of these plans is contingent on vendors providing provisions for *compensating* the effects of the call to their web services. Compensation mechanisms are available in many web service frameworks, e.g., BPEL [59]. This mechanism is used to specify application-specific ways of reversing completed activities, where a service invocation is compensated by invoking additional services (to be determined by the service provider). For example, a vendor that offers a pay service may provide compensation which involves updating the inventory and reversing any charges made to the client's credit card, encapsulated in a $\mathsf{cancel\_unshipped}$ service. However, once the item has been shipped, the client must pay a restocking fee, so a different service ($\mathsf{cancel\_shipped}$) must be used. Compensation might not leave the application in its original state, as some actions have irreversible side-effects, and sometimes compensation might not be available at all.

Given a violation, a *recovery plan* may involve "going back" – compensating the occurred actions until an alternative behaviour of the application is possible. For other violations, such plans include both "going back" and "re-planning" – guiding the application towards a desired behaviour. For example, if our user bought a used crib and decided to ship it using FedEx without consulting with the vendor (who only works with UPS), recovery simply means cancelling the FedEx shipping order and creating one with UPS. On the other hand, if our user purchased a used crib but later got a notice that delivery will be delayed (violating the "crib has been delivered within an acceptable time period" property), then recovery means both compensating executed activities (such as returning the used crib when it finally arrives and getting money back) and carrying out new activities (such as buying a new crib). Recovery plans are ranked based on length, as well as the cost of the compensation actions in them.

5 Assumptions

In this section, we describe and exemplify assumptions that our methodology places on the user configuration environment, vendors, and the web.

5.1 Individual user environment

In order to create and maintain effective personalized workflows, users are encouraged to create and maintain individual environment configurations. Such configurations include information about favourite sites and vendors, desired vendor policies and preferences, as well as additional configuration options. Information about favourite sites and vendors is used to discover web services that can execute part of the personalized workflow. Desired vendor policies are turned into monitors, thus enabling runtime monitoring of these policies. Finally, users can configure additional options, such as the maximum number of vendors to be displayed during workflow configuration, whether or not to enable checking vendor service invocation preconditions, as well as which policy monitors to enable.

In our example, our user prefers different online shopping sites depending on the type of product she is looking for: craigslist.org for used products, and sears.ca or toysrus.ca for new products. Our user also maintains a ranked list of preferred shipping companies, banks, etc. The following are some examples of vendor policies that our user prefers:

- P_1: Never pay cash if the vendor accepts a credit card.
- P_2: Always prefer slower but cheaper shipping to faster but more expensive.
- P_3: Whenever her credit card is charged, check back for a week to make sure that the charge went through and only once.
- P_4: Prefer to receive the merchandise first and be billed for it later.

Policies $P_1 - P_3$ are examples of behaviour that the user wants us to monitor at runtime, while P_4 is a preferred task ordering that can be used to guide the planning phase. With this in mind, users can add preferred (but not mandatory) ordering constraints, such as simple precedence and response properties, as well as occurrence properties, like the presence or absence of certain activities, to their high-level property specification. This can be done using simple templates, as in the Specification Pattern System [22].

Finally, our user indicates that the maximum number of vendors to be displayed is five, and that all policy monitors and service preconditions should be checked during runtime. Clearly, these and other assumptions should "follow the user" from a computer to a computer and from one environment to the next, so they should be naturally stored in the cloud.

```
webService _"http://example.org/pay"
  capability processPayment
    sharedVariables ?item ?creditCard ?order ?x ?y

    precondition pre_pay
      nonFunctionalProperties
        description hasValue "check that the item is still in stock and
          that the given credit card has enough credit for the transaction"
      endNonFunctionalProperties

      definedBy
        ?item memberOf Item and ?creditCard memberOf CreditCard and
          ?item[sku#inStock hasValue _boolean("true")] and
          ?item[price hasValue ?x] and
          ?creditCard[limit hasValue ?y] and
          ?x <= ?y

    postcondition post_pay
      nonFunctionalProperties
        description hasValue "check that a valid order has been placed and
          that the given credit card has been charged"
      endNonFunctionalProperties

      definedBy
        ?item memberOf Item and ?creditCard memberOf CreditCard and
          exists ?order
            (?order memberOf Order and
             (?order[sku] hasValue ?item[sku]) and
             (?order[client] equivalent ?client))
          and ?client[limit hasValue ?y - ?x]

    annotations
      compensation hasValue "if ?item[shipped hasValue _boolean('true')]
                                then cancel_shipped else cancel_unshipped"
    endAnnotations
```

Fig. 6: WSML definition of the pay service.

5.2 Vendor registry and configuration

Turning the web into services. In our workflow-based vision, users invoke web services in order to accomplish their goals, instead of browsing the web. That is, the web should be turned into a collection of such services. We assume that vendors publish the APIs for these services in a publicly accessible registry, and a search protocol is available that allows these to be queried based on their metadata. This assumption follows from the evolving standards (e.g., UDDI [58], WS-Discovery [60], etc.) regarding web service discovery.

Service specification. It is essential to have some notion of specification for services, at least to determine whether a particular service can be invoked at a particular step of the workflow and to discover services. Personal web is not unique in this challenge – good specifications are essential for creating quality web service applications under existing technologies. We think that the semantic web research community has a lot to offer on this topic.

Service interfaces are predominately specified using the Web Services Description Language (WSDL) [84], where the vendor indicates a service's URL and the syntax of its input/output messages. In this work, we need richer interface descriptions, since we also want vendors to specify service compensation and pre- and postconditions. The Web Service Modeling Language (WSML) [25] allows the specification of such interfaces, so we will use it in this work.

In WSML, services are declared using the **webService** keyword; the URI argument indicates where the service can be accessed. Each web service can declare at most one **capability**, i.e., the task that it carries out. Each capability has a **sharedVariables** block, which is used to indicate the variables that are available to the pre- and postconditions of the capability, which are defined using **precondition** and **postcondition** definitions, respectively. Each pre- and postcondition definition consists of an optional **nonFunctionalProperties** block, where the condition is described informally, and a logical expression preceded by the **definedBy** keyword, that formally defines the condition to check (which can be used to monitor the service). Pre- and postconditions about service ordering can also be specified in these blocks. Finally, since WSML does not have a specific keyword for specifying compensation, we added the definition of the service's compensation in the **annotations** block.

For example, Figure 6 shows the WSML definition of the pay service discussed in Section 4. This service can be accessed at `http://example.org/pay`, and has one capability, processPayment. The pay service has shared variables ?item, ?creditCard, ?order, ?x and ?y, as well as one pre- and one postcondition, pre_pay and post_pay, respectively. The precondition pre_pay checks that the item being bought is still in stock (represented by the expression ?item[sku#inStock hasValue _boolean("true")] in the **definedBy** block) and that the item's price is less than the available credit on the credit card (the rest of the expression in the **definedBy** block). Similarly, the postcondition post_pay has two parts, the first one checking that the pay service created a valid order (exists ?order ...), while the other – that the client's credit card was charged. The pay service is compensatable, since the vendor specified a compensation strategy in the **annotations** block: if the item has already been shipped, invoke the service cancel_shipped; otherwise, invoke cancel_unshipped.

Architectural support. Given that users define personalized workflows by "stringing" together services, determining if services are compatible is an important issue [63, 78]. There seems to be a lot of success in existing technologies for creating web service compositions: the Semantic Web community mainly relies on AI planning techniques to automatically create service compositions [34, 76]; and service compositions can also be created manually using services like Yahoo! pipes and Google App Inventor. Like these initiatives, we assume that services "talk the same language" w.r.t. input and output messages (ensured through the creation and use of service interface ontologies).

An orthogonal question is that of where the state of a workflow should be stored during execution. To simplify presentation, we assume that workflow data "lives in the cloud", freely available to any service that may need it. We also assume that the cloud deals with data management, formatting, and storage issues.

5.3 Availability of and Search for Crowd-Sourced Models

In our work, we do not expect users to directly create detailed goal models; instead, we rely on "crowd-sourcing" them from the web. To do so, we make the assumption that the web community (users and/or vendors) publishes goal models representing ways to elaborate and accomplish common goals. For example, some of these models may express the offerings of particular service providers such as the goal model for Amazon.com's services, while others are "good ideas" on how to accomplish common tasks on the web such as finding a good place to eat. We refer to these as *crowd-sourced models* – elaborations of goal models published by web users. Crowd-sourced models are *complete* if their leaf notes are queries to the registry, resulting in lists of vendors which implement the tasks described by those nodes.

We envision the crowd-sourced goal models to be another resource type alongside HTML documents, images, etc. that can be published on the web and be accessible via web search engines – we call such a search engine *super-google* later in this paper. The search uses community rankings of the quality of the model together with information available in the user context (see Section 5.1) and individual vendor contexts to discover most suitable models.

We also assume the availability of a *common ontology* which represents a set of concepts within our domain, (i.e., the online shopping domain) and the relationships between those concepts. It plays a pivotal role in unifying the terms in different contexts and in dealing with potential differences in levels of abstraction as well as any inconsistencies. For example, using ontological relationships between words (e.g., the ontology provided by WordNet [26]), one can infer that a crib is a type of furniture. Thus, super-google can search not only for crib buying scenarios, but also how to buy furniture.

6 Step 1. Elicitation

Elicitation refers to the activities concerned with understanding the personal web user's objectives and preferences and expressing them in a suitable notation. The intended outcome of the elicitation phase of PWWM is a goal model that covers: (1) the user's high-level goals, (2) alternative means for realizing the goals, and (3) the main selection criteria for alternatives. Additionally, the user can provide information on how each of the alternatives are evaluated against her selection criteria.

We assume that a typical user might be unable to construct goal models directly, and we use a simplified approach for eliciting high-level goals. First, the user is asked to describe the web transaction workflow using natural language, and lexical analysis based on keyword search is used to extract high-level goal model elements from the description. The underlying justification for keyword search is that *a goal* is a statement of intent. Table 1, adapted from [83], lists several useful goal-related keywords that might drive goal search in the early stages of elicitation. Then, a wizard is used to elicit elaborations of these high-level goals by asking the user to state "how" to achieve the goals. For example, if a goal is to "have a place for baby to sleep", then asking "how?" might yield alternative approaches such as "Arranging bed share" or "Buying a crib". This process is iterated on these subgoals until the user can no longer elaborate them.

The selection criteria such as "should be inexpensive" are also elicited using a wizard and added to the goal model as soft goals. Such criteria constitute *user preferences* – desires that are not mandatory but nice-to-have when achieving the mandatory goals. Some of these are goal-independent and are described in Section 5.1, whereas others are elicited just for the current task. Focusing on OR-decompositions, the user assesses the impact of each alternative on each of the identified criteria, wherever applicable. Well-known priority elicitation techniques (see more about them in Section 7.4) can be used to both quantify the impact of alternatives to preferences and (if this is desired at this early stage) to acquire a general sense of which of the criteria are by default

Prescriptive	shall, should, must, has to, to be, may never, may not, should never, should not
Intentional	in order to, so as to, so that, objective, aim, purpose, achieve, maintain, avoid, ensure, guarantee, want, wish, motivate, expected to

Table 1: Useful keywords for goal search.

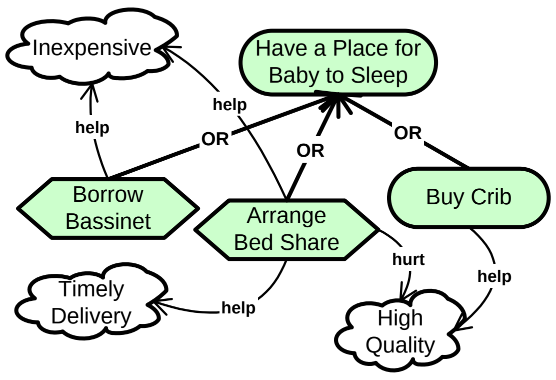


Fig. 7: User's initial goal model for the crib purchasing example.

more important to the user. These preferences pertain to the general characteristics of preferred workflows (rather than particular preferred vendor instances such as Sears vs. Walmart), and thus we refer to these as *workflow-level preferences*. They get updated and become more specific as the model is further elaborated.

Thus, the result of the elicitation process is a well-formed goal model containing the high-level expression of the user's objectives for the web transaction, as well as some general preferences as to what is important for the user. For our crib buying example, we synthesize the goal model (Figure 7), capturing the high-level alternatives suggested by the user for handling the crib shopping scenario. The soft-goals that appear in that model constitute initial expressions of preferences, subject to prioritization in later stages.

7 Step 2. Goal Refinement

The objective of the Goal Refinement step, depicted in Figure 8, is to enable the user to elaborate the high-level goal model produced in Step 1 by showing how these goals can be refined into lower-level ones and, conversely, how lower-level goals contribute to higher-level ones. The refinement continues until the model is *complete* (see Section 5.3), by iterating through the following steps:

1. **Step 2.1 (Search):** The user selects a high-level un-operationalized goal and searches the web for a list of crowd-sourced models that can elaborate and/or operationalize the selected goal. She then goes through the list to pick and download one of the crowd-sourced models that best fits her expectations.

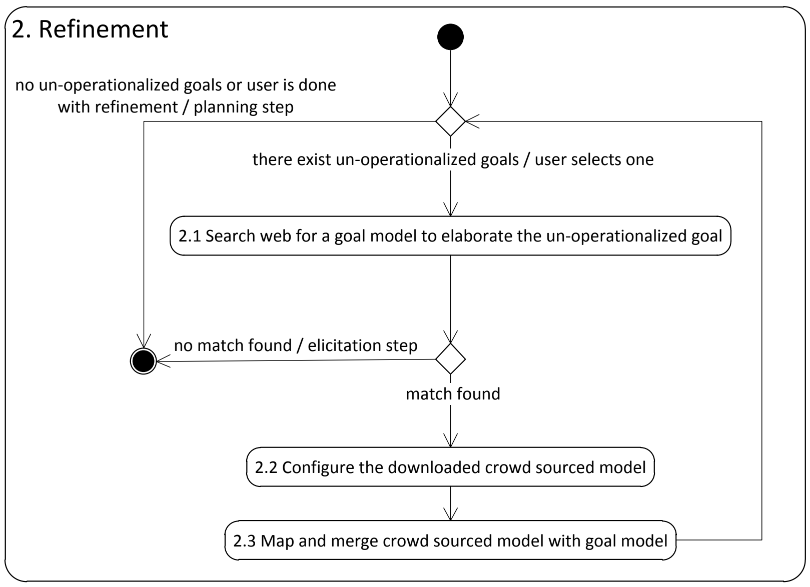

2. Refinement
no un-operationalized goals or user is done with refinement / planning step
there exist un-operationalized goals / user selects one
2.1 Search web for a goal model to elaborate the un-operationalized goal
no match found / elicitation step
match found
2.2 Configure the downloaded crowd sourced model
2.3 Map and merge crowd sourced model with goal model

Fig. 8: A Refinement step of PWWM.

2. **Step 2.2 (Configure):** Subsequently, the user may modify and configure the downloaded crowd-sourced model to customize it to her specific needs.
3. **Step 2.3 (Match and Merge):** Then, she attempts to integrate the crowd-sourced model and her initial goal model, developed during elicitation (see Section 6). This, in turn, involves finding a mapping between the crowd-sourced and the initial goal model, and then merging the two models.
4. **Step 2.4 (Preference Refinement):** Finally, the user refines her original general preferences by adding more detail and by defining priorities among them.

If the search in step 2.1 fails to produce satisfactory results, the process shifts back to the elicitation step to get the user's assistance in revising the selected node. The refinement process is repeated as long as there are un-operationalized nodes that the user wishes to elaborate and/or operationalize. Also, in principle, the user can take the crowd-sourced model and add her own tasks to it, which means that she may need to iterate over steps 2.1 and 2.2 multiple times. In the remainder of this section, we discuss each of the activities in more detail.

7.1 Searching for crowd-sourced models

In this step, the user first chooses one of the un-operationalized goals in her initial goal graph. For our example in Figure 7, among the three proposed alternative for handling the baby's sleeping place, the user chooses the "Buy crib" alternative and concentrates on refining that particular goal. She then attempts to find a crowd-sourced model describing how that goal can be decomposed into smaller steps, and what alternative scenarios exist on the web to carry out it.

The search engine – super google – returns a ranked list of crowd-sourced goal models describing how the "Buy Crib" process can be carried out on the web. The user

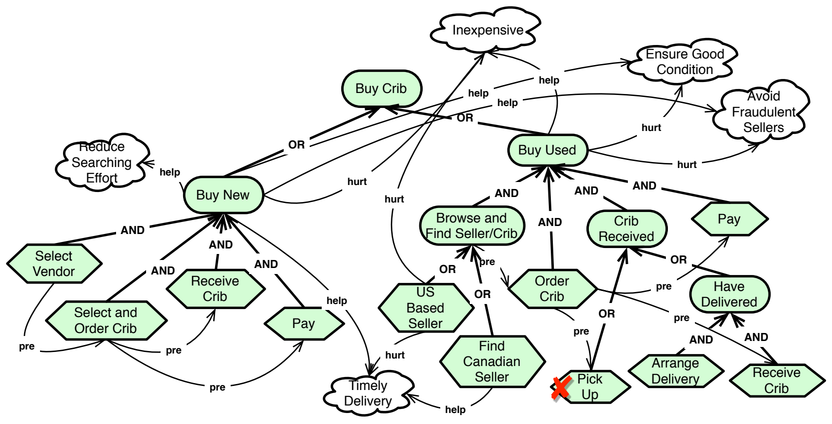


Fig. 9: The crowd-sourced goal model for buying a physical large item online in North America. The process "pick up" is removed by the user during configuration (Step 2.2).

has the option of going through the list and reviewing the ratings and comments to make her final decision. In our example, the final crowd-sourced goal model chosen by the user is shown in Figure 9. As shown in the figure, the crowd-sourced model suggests to decompose the "Buy Crib" goal into two main subgoals: "Buy New" and "Buy Used". Each subgoal is then decomposed into sequences of tasks. Specifically, the "Buy New" is decomposed into the sequence of tasks: "Select Vendor", "Select and Order Crib;", "Receive Crib", and "Pay". The "Buy Used" goal is decomposed into a similar sequence, except that there are two options for finding a vendor, since the user is free to choose a vendor from Canada or the US, and the user can receive the crib either by picking it up herself or by arranging it to be shipped to her place.

7.2 Configuring found models

The crowd-sourced goal models are generic descriptions with several alternatives and thus are highly configurable. Of course, not all alternatives are applicable to all users. Hence, we expect the user to configure the crowd-sourced goal model based on her needs and according to her personalized scenario, and then integrate it with her personal model.

For example, the user may remove the "Pick up" alternative from the goal model in Figure 9 because she does not have a car and therefore cannot pick up the crib herself. While not illustrated in our example, the configuration could be more advanced. In particular, it could involve choosing values for a number of configurable parameters, e.g., the shipping insurance amount if the shipment is to be insured. Also note that the user may decide to extend the crowd-sourced goal model by adding her own tasks to it. In our example, the user may add an "ensure partner agreement before a purchase is made" task, involving sending an email to her spouse and awaiting a confirmation.

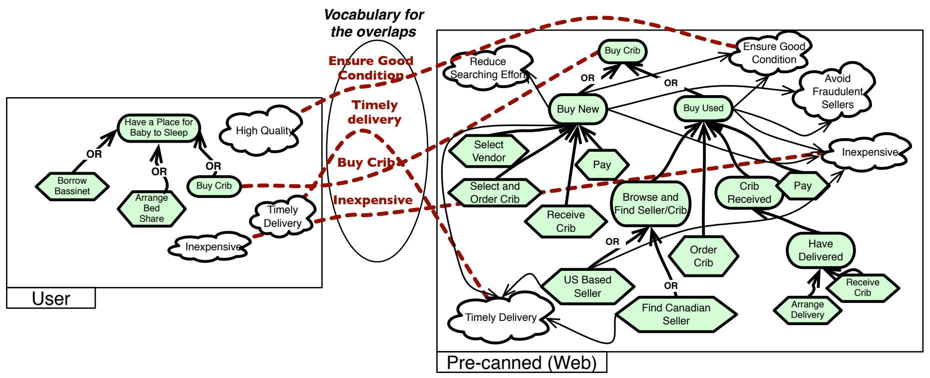


Fig. 10: Relationship between the personal goal model in Figure 7 and the crowd-sourced one in Figure 9.

7.3 Mapping and merging of goal models

Once the crowd-sourced goal model has been configured, it needs to be merged with the user's (personal) goal model. This in turn requires the relationship between the two models to be specified. When the models are developed in a centralized manner, the relationship can be left implicit and defined through conventions, e.g., name equivalence if models have a common vocabulary, or identifier equivalence if models have common ancestors. In the context of personal web, it is very hard to put such conventions in place as the models are developed independently and often without any prior coordination. The relationships between independently-developed models have to be specified explicitly instead [16]. These relationships are often established through a combination of manual and automated matching based on heuristics [57].

In our example, both the personal and the crowd-sourced models are small, and the matching can be done manually. Figure 10 shows the relationship that the user has defined between hers and the crowd-sourced model. The relationship defines the overlaps between the concepts in the two models: "Buy product" is mapped to "Buy Crib". "Get in a definite time" is mapped to "Timely delivery". "Inexpensive" to "Inexpensive". "Ensure good quality" to "Ensure good quality". The relationship is expressed as a set of labelled mappings between concept pairs. The labels on the mappings specify the vocabulary that should be used for the shared concepts in the merge. The result of merging the personal and the (configured) crowd-sourced models is shown in Figure 11.

Alternatively, we can use automated matching techniques when the models are large or when the user is not certain about her manually built relationships and would like the system to provide her with some recommended matchings. Our automated matcher uses the common ontology discussed in Section 5.3 to unify the terms in different models and discover potential matches in a similar way that the common ontology can be used by super-google for searching the web.

7.4 Preference Refinement and Prioritization

As shown in Figure 11, the merged goal model includes *workflow preferences* – things that the user generally likes to see satisfied – in the form of soft goals. Examples of

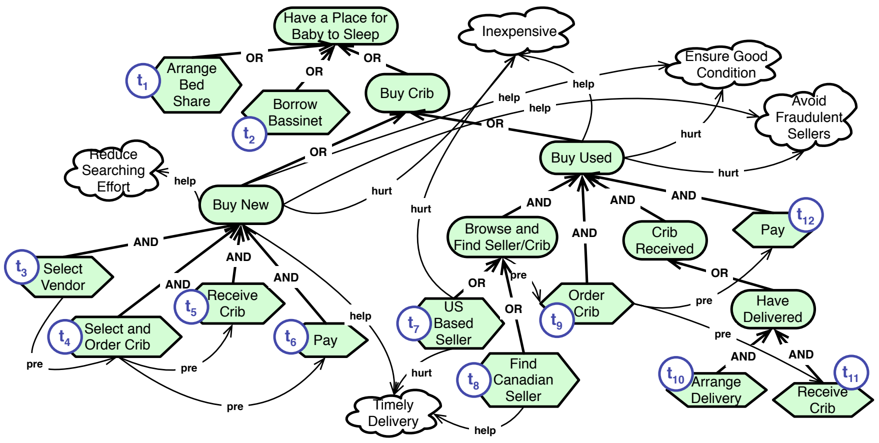


Fig. 11: Merge of the personal goal model in Figure 7 and crowd-sourced one in Figure 9 with respect to relationships in Figure 10.

these are "Ensure Good Condition", or "Inexpensive". In addition, while the model in Figure 11 does not prescribe whether payment (task t_{12}) precedes delivery (task t_{11}), the user may want to express preferences pertaining to the ordering of tasks. In Section 5.1, such preference was given using P_4, indicating that shipping should precede payment, if possible.

The goal model together with those preferences constitute a relatively stable representation of stakeholder desires and alternative requirements. In practice, in different situations or based on new information that arrives while our user already attempts to fulfill her goal, the relative importance of preferences changes. Specifying *priorities* amongst user preferences allows us to describe their relative importance at a given point in time. These come in the form of weighted linear combinations of individual preferences – the weight being a measure of the relative importance of the corresponding preference. The linear combination can then be seen as an *objective function* to be optimized: from the large number of plans that are implied by the goal model, we are interested in those that satisfy as many of the important preferences as possible.

Elicitation of the weights is possible through a variety of techniques from simple ad-hoc assessment (e.g., [86]) to methods based on pairwise comparison, such as the Analytic Hierarchy Process (AHP) [2, 40, 67]. Back to our example, assume that an "Inexpensive" purchase is "strongly more" preferred to "Reduce Searching Effort". Applying AHP, which involves assigning the corresponding preference values to a comparison matrix and estimating its eigenvector, gives us priority weights 0.83 and 0.17, respectively. In practice, a preference profile can include more than two components and more detailed relationships between them, e.g., vendor-level preferences like "Use services by Speedy Delivery Inc." or even vendor-specific temporal properties like "If you purchase from Maple Cribs Inc., use their delivery service as well".

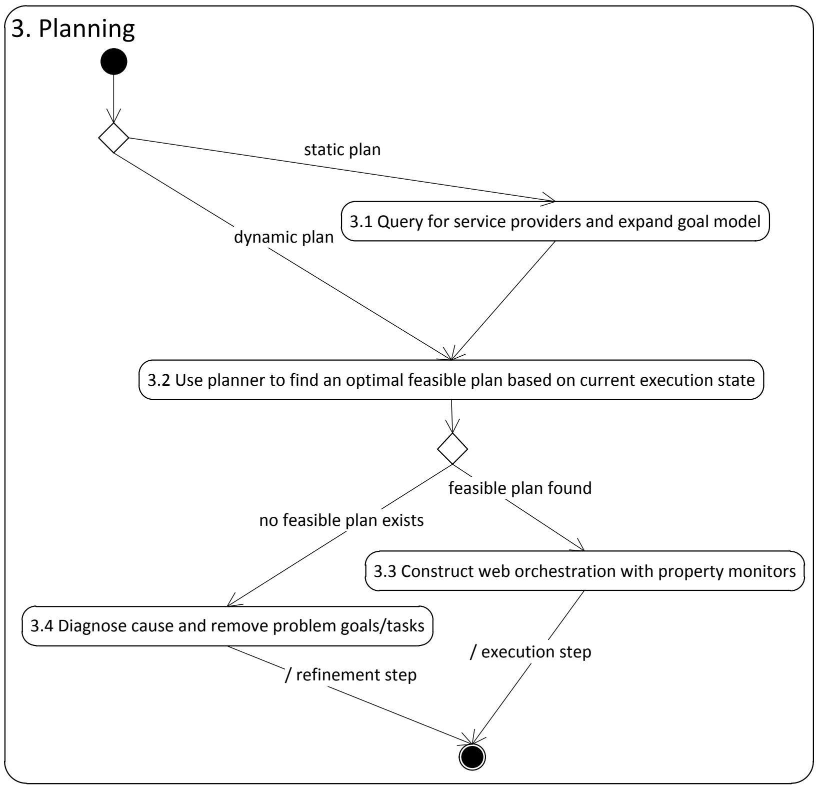

3. Planning
static plan
dynamic plan
3.1 Query for service providers and expand goal model
3.2 Use planner to find an optimal feasible plan based on current execution state
feasible plan found
no feasible plan exists
3.3 Construct web orchestration with property monitors
3.4 Diagnose cause and remove problem goals/tasks
/ execution step
/ refinement step

Fig. 12: A Planning step of PWWM.

8 Step 3. Planning and Web Service Creation

During this step, state-of-the-art preference-enabled planning algorithms [10, 75] read the goal models and preference prioritization – automatically translated into a planning specification language – and compute ranked plans within the detailed goal model that satisfy the customer's prioritization, by means of maximizing the given priority as expressed in the value of an objective function (see Section 7.4). Picking plans lower in the ranking indicates relaxing preferences. If the user does not like any of the produced plans, she may want to change the preferences, resulting in the planner computing a different set of plans.

Recall that a detailed goal model is fully operationalized if its leaf tasks contain queries allowing us to discover appropriate web service calls that can execute them. Models get fully operationalized in the Refinement step. The Planning step allows finding web services before the plan is generated (*static planning*) or as the plan is being executed (*dynamic planning*). The advantage of a static plan is that it guarantees that vendor policies, expressed through the pre- and postconditions of their service invocations, are taken into account. However, compared to dynamic planning, it is much more expensive for the planner to generate and also does not give the user a complete flexibility in service provider selection. In either case, failure to complete execution of the plan may result in having to change the vendors while the actual plan stays the same, or in relaxing or changing the the workflow-level preferences to get a new plan, or, less likely but possibly, going back to the refinement step in order to update the goal model itself.

In the rest of this section, we provide more detail on static vs. dynamic planning, describe how each method allows for creation and composition of actual services, and how runtime monitoring is possible and beneficial in each case.

8.1 Static plan and web service creation

Step 3.1. Query for service providers and expand goal model. The *static* approach to generating a plan requires that the leaf-level tasks of the goal model be expanded with lists of providers offering the service of interest. Thus, we begin by executing the service registry query associated with each task of the operationalized goal model. The outcome of the query returns a number of services (the maximum number can be controlled by the user – see Section 5.1), with a potential (crowd-sourced) ranking of how well they perform the service in question. Some of the services may also explicitly specify their precise pre- and postconditions, as offered by the providers themselves (see Section 5.2).

For example, consider generating a static plan for the goal model in Figure 11. First, we query the service registry to produce a list of particular services offered by providers to accomplish these goals. For example, one of the services associated with task t_9 is a placeOrder service offered by a Canadian crib vendor, Maple Cribs Inc. This service has a precondition (P_5) that the selection service offered by the same vendor (selectByType) must be invoked first. This vendor also offers a delivery service, arrangeDelivery, which presumes that their service pay, associated with task t_{12}, has been performed. In other words, the arrangeDelivery service has the following precondition (P_6): “pay precedes arrangeDelivery”. The arrangeDelivery service also has a postcondition (P_7): “the user should eventually receive the item (receiveDelivery)”.

The same tasks can also be accomplished by the corresponding services of a US-based company, Rock Baby Rock Ltd. However, their delivery service (accomplishing t_{10}) presupposes the use of their own order service (associated with t_9). This is also a service precondition (associated to the delivery service): the Rock Baby Rock order service must be invoked before its delivery service can be used.

Step 3.2. Static plan generation. Given the expanded goal model produced as a result of Step 3.1, the planner can readily find sequences of steps based on concrete services that vendors provide. Moreover, the fact that a plan is found *guarantees* that there exists a service composition that satisfies the user’s goal – at least if provider-specified pre- and postconditions are complete and correct. Furthermore, the user-maintained vendor-specific preferences, if any, can also be used by the planner to produce rankings or service composition possibilities.

If the planner fails to find a plan with higher ranked vendors it will attempt to find one with lower ranked vendors, which may correspond to the same sequence of requirements-level tasks but with different service bindings. Alternatively, the user may change her preferences, resulting in the planner calculating new rankings. Either way, while the exact choice of vendors used in the resulting service composition is *affected* by the user, it is not fully *controlled* by the user.

Returning to our crib-buying example, the planner can produce a number of plans which satisfy the vendor pre- and postconditions while taking user preferences into account. If it is more important to pay after delivery (see P_4 in Section 5.1), the combined order+delivery package offered by Rock Baby Rock may be unavoidable. If a higher preference is given to a Canadian vendor (or to Maple Cribs specifically, reflecting a pre-existing vendor-specific preference resulting from an earlier crib purchase), a delivery service can still be arranged through a third party since the use of the Maple Cribs ordering service does not require that their own delivery service is used as well.

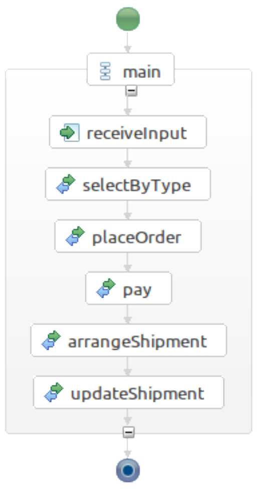
Assume that the user's preference profile is "Inexpensive [Crib]" (with weight 0.5) "Use services by Speedy Delivery Inc." (weight 0.3) and "If you purchase from Maple Cribs Inc., use their own delivery service as well" (weigh 0.2) – in practice, preference profiles can be much richer than this one. The following are the three top scoring statically generated plans that use Maple Cribs Inc services (score value in parenthesis):

$sp_1(0.8)$ = {selectByType, placeOrder, pay, arrangeShipment, updateShipment}
$sp_2(0.8)$ = {selectByType, placeOrder, pay, arrangeShipment, receiveDelivery}
$sp_3(0.7)$ = {selectByType, placeOrder, pay, arrangeDelivery, receiveDelivery}

Services selectByType, placeOrder, pay, arrangeDelivery, receiveDelivery are offered by Maple Cribs Inc, whereas arrangeShipment and updateShipment are offered by Speedy Delivery. For example, in plan sp_1, the user buys the crib from Maple Cribs Inc., but ships it with Speedy Delivery since it offers a better delivery experience than Maple Cribs Inc and thus occurs in the user's preference profile with a higher weight (0.3) than shipping with Maple Cribs (0.2).

Step 3.3. Web orchestration and property monitors (static). Since a static plan is just a simple sequential orchestration of web services, we can make it executable using BPEL. Figure 13a shows the BPEL implementation of the top-ranked plan sp_1. The workflow begins with the receiveInput activity, which stores the workflow input parameters on the cloud in order to make them available for other services (as discussed in Section 5.2). Each task in plan sp_1 is carried out by invoking a web service (the corresponding activities in the BPEL diagram are preceded by a symbol). We attach compensation handlers to the activities that invoke compensatable services (not visible in the BPEL diagram). Figure 13b shows an example of a BPEL compensation handler – the one attached to the pay service invocation. As indicated in Section 5.2, the pay service is compensated by executing the cancel_unshipped service, since the item is paid for before shipping in plan sp_1. Finally, since we assumed that services "speak the same language" w.r.t. input and output messages (see Section 5.2), we do not deal with data management/formatting/storage issues which exist between today's web services.

Since this orchestration can fail at runtime, at this point we also generate monitors for this orchestration. For statically-generated plans, these monitors come from a number of sources which we describe below. Note that the event receiveInput (from Figure 13a) does not appear in any of the monitors because it is a BPEL <receive> activity, indicating the beginning of the workflow.



(a)

```
<scope name="pay_scope">
   <invoke ... operation = "pay" ... />
   <compensationHandler ... >
       <invoke ... operation = "cancel_unshipped" ... />
   </compensationHandler>
</scope>
```

(b)

Fig. 13: (a) Static BPEL implementation of plan sp_1 (see Section 8.1) and (b) BPEL compensation handler for pay invocation.

1. User workflow-level preferences, including high-level order and occurrence properties. These are used during construction of a static plan but the user may choose to register for the corresponding monitors anyway, to check for runtime failures. For example, some monitors corresponding to user preference policies P_1 and P_4 (see Section 5.1) are shown in Figure 14. The monitors for P_2 and P_3 are very similar to M_1 and thus are not shown. In the case of policy P_1, we first check if the chosen vendor supports both cash and credit actions, i.e., we check whether $\{\mathsf{cash}, \mathsf{credit}\} \subseteq \Sigma_v$ is true, where $\Sigma_v \subseteq \Sigma$ is the set of actions offered by the vendor. If so, then payment using cash is a forbidden behaviour (pay followed by cash), and leads to the bad state 3. On the other hand, payment via credit (pay followed by credit) is a desired behaviour, leaving the monitor in a good state 4. Monitor M_4 checks that the user receives the item before paying for it (leaving the monitor in the good state 4). If payment occurs before the user receives the item, the monitor ends up in the bad state 2, indicating a violation.

2. Vendor-specified pre- and postconditions and expected workflows. Service providers may assume that their services are invoked in a particular order, or work with others in a particular way. While these workflows are used in static plan construction, monitors can still check if stated postconditions achieved by individual invocations hold, or whether various failures affected the expected vendor workflow. For example, since plan sp_1 uses Maple Crib's placeOrder service, precondition P_5 is turned into a monitor (see

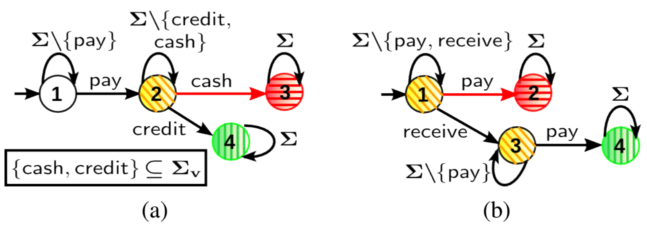


Fig. 14: User preference monitors: (a) M_1 and (b) M_4, corresponding to policies P_1 and P_4 (see Section 5.1), respectively.

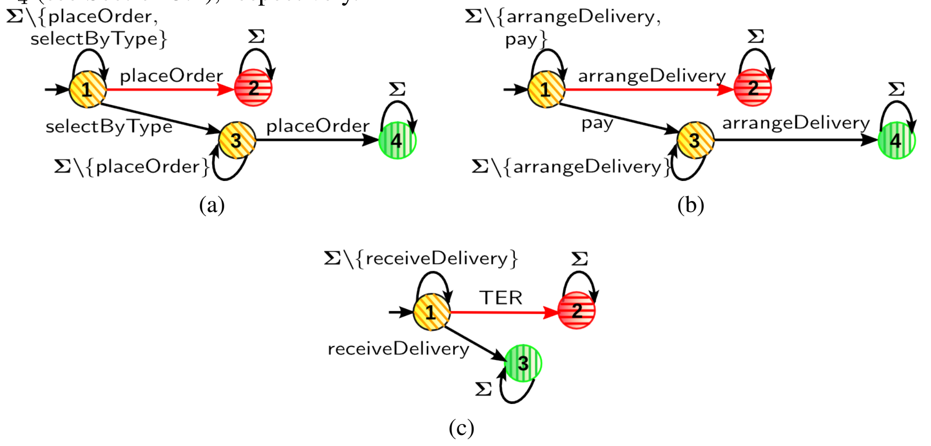


Fig. 15: Vendor monitors: (a) M_5, (b) M_6, and (c) M_7 corresponding to preconditions P_5, P_6 and postcondition P_7 (defined earlier in this section), respectively.

Figure 15a). Since plan sp_1 does not use Maple Crib's arrangeDelivery service, we do not add the monitors M_6 and M_7 (see Figure 15) to the set of active monitors. Finally, plan sp_1 also invokes the pay service defined in Section 5.2, so the pre- and postcondition expressions specified in the WSML file become assertions that should be checked before and after service invocation, respectively.

3. User goal models produced by the Refinement step. There are many reasons why a started plan does not finish, mostly due to the internet being unreliable and/or failure of individual vendor services. At runtime, we aim to check that the entire chosen plan completes successfully. For example, if plan sp_1 runs to completion, monitor M_8- in Figure 16 is left in a good state 6 (coloured green and shaded vertically). On the other hand, if the workflow unexpectedly terminates at any step of the plan, the monitor ends up in a bad state 7 (coloured red and shaded horizontally).

4. User stories from crowd-sourced models. Crowd-sourced models used during the Refinement step may optionally come with user stories, positive or negative, e.g., some users reported that Maple Cribs Inc. products shipped with Speedy Delivery arrive in bad condition. This property is checked using M_9, shown in Figure 17: the monitor is left in a bad state 3 if arrangeShipment (Speedy Delivery's service) is invoked after placeOrder (Maple Cribs' service). Our methodology allows users to register moni-

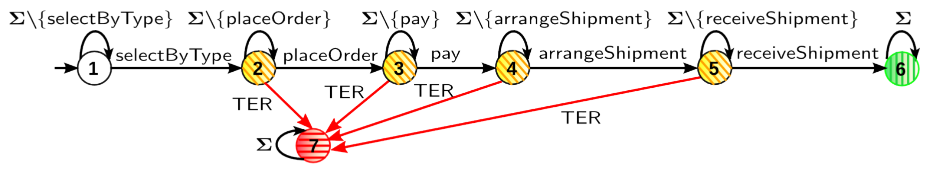


Fig. 16: Monitor M_8 to check that plan sp_1 (Section 8.1) is run to completion.

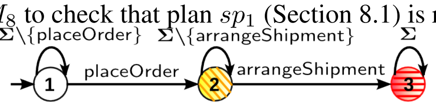


Fig. 17: Monitor M_9 for checking a negative user story described earlier in Section 8.1.

tors to check whether their own workflows are subject to such desired or undesired behaviours.

8.2 Dynamic plan and web service creation

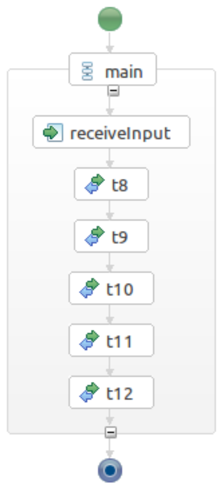
Step 3.2. Dynamic plan generation. The second approach to planning assumes dynamic task-to-service binding. The planner generates a sequence of abstract requirements-level tasks that optimize user preferences. At runtime, a post-processor queries the service registry to find different service providers that implement the current step of the plan. The user chooses one from the suggested set to call. Since each choice of service providers is done “greedily”, there is no guarantee that the resulting composition is feasible, and verifying this is deferred to the monitoring component. Failure to fulfill a plan does not necessarily imply the need to choose a less preferred one or re-planning, but may involve trying different task-to-service bindings, through querying the repository again. Compared to statically-generated plans, this approach is computationally cheaper (on the planner) and gives users more control in the process of choosing their preferred set of vendors. However, the likelihood of the initial failure and the need to try the process multiple times increases.

Let us return to the example of Figure 11. Assume that the preference profile includes the quality preferences “Inexpensive” and “Timely Delivery” as well as the temporal preference “Pay after Delivery”, with weights 0.4, 0.4 and 0.2, respectively. The planner generates two top-ranked plans:

dp_1 = $[t_8, t_9, t_{10}, t_{11}, t_{12}]$ with score 1.0 (optimal)
dp_2 = $[t_8, t_9, t_{12}, t_{10}, t_{11}]$ with score 0.8

Thus, the highest ranked plan allows for both an inexpensive purchase and a timely delivery, as it allows buying a used crib from a Canadian seller. It also prescribes that payment must happen after delivery. Thus, all components of the preference profile are satisfied. The second plan satisfies the first two components but not the third one, hence the lower score.

Both plans are descriptions of desired workflows at a high level, without any information about the particular services that will implement it. In what follows, we assume that the user picked plan dp_1 to execute.



(a)

```
<partnerLink name="t12_service" partnerLinkType="services:payService"
  myRole="PayServiceRequester" partnerRole="PayServiceProvider"/>

<partnerLinkBinding name="payService">
  <property name="wsdlLocation">payService.wsdl</property>
</partnerLinkBinding>
```

(b)

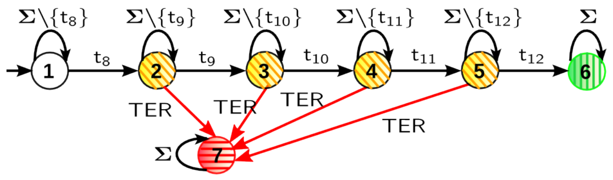
```
<service name="MapleCribsInc">
 <port name="payServicePort" binding="tns:payServiceBinding">
  <soap:address location="http://maplecribs.ca/pay"/>
 </port>
</service>

<service name="RockBabyRock">
 <port name="payServicePort" binding="tns:payServiceBinding">
  <soap:address location="http://rbr.com/pay"/>
 </port>
</service>
```

(c)

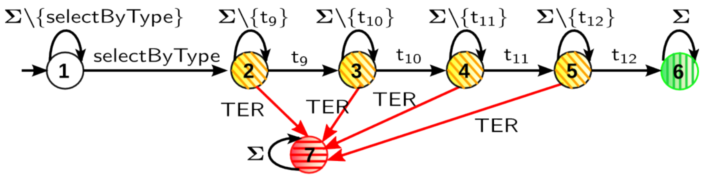
Fig. 18: (a) Dynamic BPEL implementation of plan dp_1, (b) partner link pointing to a generic t_{12} service, and (c) snippet of the WSDL file where the concrete pay services are defined.

Step 3.3. Web orchestration and property monitors (dynamic). The dynamic web orchestration used by this approach to planning requires explicating queries to the service registry in order to find appropriate bindings at each step of the plan.

While BPEL engines augmented with aspects [4] can be used for implementing this approach, there are provisions to do this in native BPEL as well, which we follow here. In BPEL, services are made available through partner links which use the information in the referenced WSDL definition files to determine which services are available. BPEL supports dynamic binding of partner links, making it possible to modify various partner link parameters, like service URIs (host, port and path) and target service names at runtime. This means that dynamically generated plans can also be implemented using



(a)



(b)

Fig. 19: Monitor M_{10}: (a) initial version where no tasks are bound to services, and (b) after user picks service selectByType to realize task t_8.

BPEL, as long as the concrete services are defined in the linked WSDL files. The contents of the WSDL file for each dynamic binding is generated right before the binding is used. Compensation is defined the same way as for static plans.

For example, the BPEL implementation of plan dp_1 is shown in Figure 18a, where all invocation activities point to generic services instead of concrete services. The partner link for the generic t_{12} service is shown in Figure 18b, referring to the WSDL snippet in Figure 18c where two concrete pay services, offered by Maple Cribs Inc. and Rock Baby Rock Ltd, are defined.

When the user executes the BPEL orchestration in Figure 18a, the runtime environment first queries the service registry to find candidate services for activity t_8. The query results in several Canadian and US vendors, including Maple Cribs Inc. and Rock Baby Rock Ltd. We assume that vendors can be ranked in a variety of ways, e.g., using the notion of crowd-sourced "quality" of a vendor or the user's personal vendor-specific preferences. At each point in the execution, the runtime environment maintains the current "state" of the system and thus displays only those vendors whose pre-conditions satisfy this state.

Unlike the static case where all monitors are generated before execution begins, here monitors are generated on-the-fly as plan steps become operationalized. For example, Figure 19 shows two versions of monitor M_{10}, which checks that plan dp_1 defined earlier in this section is run to completion. The transition labels of the initial monitor (Figure 19a) are placeholders, set as the user chooses particular services. Suppose the user selects Maple Cribs Inc. and attempts to execute service selectByType provided by this vendor (which we suppose includes browsing products, adding them to a cart, etc.). Monitor M_{10} is updated to reflect this choice, resulting in the monitor shown in Figure 19b. In addition, vendor-defined pre- and postconditions are turned into *automatically-registered* monitors since these are no longer satisfied by plan construction. Otherwise, the sources of monitors are the same as discussed in the static case but registered and invoked on-the-fly.

Once the order is placed (t_9), the user proceeds with arranging a delivery (t_{10}). The service registry is queried again, this time returning two alternative services: arrangeDelivery and arrangeShipment, offered by Maple Cribs Inc and Speedy Delivery, respectively. Our user decides to keep shopping with Maple Cribs Inc., so monitor M_6 (see Figure 15b) is added to the set of active monitors. This monitor is immediately violated, since Maple Cribs' pay service has not been invoked on this execution trace. The run-time environment notifies the user that a monitor violation occurred, prompting her to pick an alternative service for t_{10} (e.g., arrangeShipment) to continue executing dp_1.

The stepwise find-and-execute process described above continues either until all the tasks in the plan are performed (success), or until no services satisfying the existing state of the system can be found (failure). In the latter case, a *recovery process* is initiated (see Section 9), allowing the user to try to execute the same plan but with a different choice of vendors. For example, she may want to withdraw her order from Maple Cribs Inc. if she cannot find an affordable delivery option later on.

In the end, if suitable bindings are not found, the user may choose to relax her preferences and move on to the next plan in the ranking or change her preferences and replan, effectively choosing in both cases a different general workflow and start querying for services step-by-step all over again (see Section 8.3).

8.3 Step 3.4. Replanning

Replanning happens if the user changes her preferences. This step can be entered both from the plan generation step (Step 3.2) and from the execution step (Step 4). In the latter case, the current state of the plan being executed becomes another input to the planner, to give higher rank to those plans that include already executed steps.

Returning to the example of Figure 11, assume that the user follows the dynamic planning approach having the preference profile "Inexpensive" (0.4), "Timely Delivery" (0.4) and "Pay after Delivery" (0.2). While the plan $[t_8, t_9, t_{10}, t_{11}, t_{12}]$ is optimal for this profile, suppose the user consistently fails to find a suitable binding allowing her to pay after delivery, as the plan requires. Some time passes; she becomes increasingly more impatient and willing to pay more just to finish her purchase. She thus updates her preference profile, adding "Reduce Searching Effort" as a relevant and important goal, with weight 0.5, while the weights of all other preferences are reduced to half their original ones. Without knowing the current state of execution of the user's original plan, the planner would suggest a brand new plan $dp_3 = t_3, t_4, t_5, t_6$ involving purchase of a new crib. However, if some steps of the original plan have already been performed, e.g., placing an order on a used crib as a result of executing t_8, they would now need to be cancelled, and compensation for t_8 – returning the crib – contributes negatively to the "Reduce Searching Effort" goal; thus, the planner will consider alternative plans, some of which involving getting a used crib (but paying before the delivery).

9 Step 4. Execution and Recovery

The Execution step allows the user to register a number of monitors and then run the generated BPEL, executing the plan step-by-step and updating the states of all the registered monitors until one of the following events happens: (a) some monitor fails –

at which point PWWM starts a *recovery step*; (b) the user decides to change her preferences (e.g., because the next step of the dynamic plan does not yield any service provider choices) – at which point PWWM enters Step 3.4; (c) the complete plan succeeds, satisfying the goals of the user – at which point PWWM concludes successfully; and (d) the user abandons her plan altogether and decides to start again, e.g., with the Elicitation phase.

The Recovery step uses semantic information about services involved in the workflow to attempt to fix the problem discovered with the orchestration using runtime monitoring. We explored such *property-guided recovery* in the context of traditional web applications in [72, 73], where both the orchestration and its properties are defined by the application developer, but recovery plans are computed for individual execution traces. The recovery process is easily adapted to reasoning about personal web, as we illustrate below.

We discuss handling static and dynamic plans separately.

Execution and Recovery: Static Plans. For the static plan, execution just involves running the generated BPEL orchestration. The only monitors activated by default are those that check that the entire plan executes successfully. Other monitors, such as those checking for positive or negative user stories obtained from the web, or checking vendor workflows, user preferences or vendor pre- and postconditions can be activated optionally.

For example, if our user decides to execute the static plan sp_1 defined in Section 8.1, only monitor M_8 (see Figure 16) is automatically added to the set of active monitors. Violations of this monitor indicate that the chosen plan could not be executed to completion. Monitors $M_1 - M_4$ (see Figure 14), as well as monitors $M_5 - M_7$ (see Figure 15), corresponding to user preferences and pre-, postconditions, respectively, represent properties that were taken into account during plan generation and are thus satisfied by construction. However, the user could still decide to register these monitors, since physical problems, like a server crash, can affect the outcome of the selected plan. Finally, the user selects whether or not to register monitor M_9 (see Figure 17), corresponding to a crowd-sourced user story.

Suppose the user is executing the static plan sp_1. She successfully interacted with Maple Cribs' services (leaving monitor M_8 in state 4), and she now invokes the arrangeShipment service provided by Speedy Delivery. However, a power outage in Ottawa knocked Speedy Delivery's data center off the grid, and PWWM timed out (sending a TER event) while waiting for arrangeShipment to respond. This leaves M_8 in the bad state 7, signalling that sp_1 could not be completed; thus, PWWM attempts to recover from this error.

We cannot modify statically created plans, since we do not know how these changes affect all the constraints taken into account when generating the plan. So recovery entails getting the user to try a lower-ranked static plan. In our example, the next best ranked plan was sp_2; however, sp_2 also invokes arrangeShipment and so may not be a good recovery plan candidate. The next plan, sp_3, while ranked the lowest, does not invoke arrangeShipment. It also has the same first three steps as sp_1. Picking this plan during recovery entails minimal compensation, making it an excellent candidate. If none of the statically computed plans can replace the current one (according to the user), she needs to change her preferences or return to te Elicitation phase, to generate new static plans.

Execution and Recovery: Dynamic Plans. Execution of dynamically generated plans entails running a query to the service registry, getting the user to choose among the list of potential service providers and then continuing. In addition to a number of monitors created to make sure that the plan is executed successfully, dynamic planning also includes activating, at runtime, monitors which check that pre- and postconditions of the user-chosen service providers are correctly satisfied. And, as in static plans, the user may optionally decide to invoke monitors to check for positive or negative user stories which are obtained from the web and associated with a particular service provider they chose to use.

For example, suppose our user decides to execute the dynamic plan dp_1 defined in Section 8.2. Thus, the monitors $M_1 - M_4$ and M_{10} are automatically registered. Monitors $M_5 - M_7$, on the other hand, are registered only if the associated service is invoked. As in the static case, the user can choose whether or not to register M_9.

During execution, suppose the user picked services selectByType and placeOrder to realize tasks t_8 and t_9, respectively (leaving monitor M_{10} in state 3). Since she picked placeOrder, monitor M_5 is also registered and then updated to reflect the current execution trace. This leaves M_5 in the good state 4, which means that placeOrder's precondition is met by the current execution trace. Also, since state 4 of M_5 is a sink state, this monitor can now be unregistered. In the next step, the user must pick a service to operationalize task t_{10}. She decides to use Maple Cribs' arrangeDelivery service, so monitors M_6 and M_7 become registered. Again, we update the state of these new monitors using the current execution trace, so M_6 (M_7) is left in state 2 (1). State 2 of M_6 is bad, indicating that arrangeDelivery's precondition does not hold on the current execution trace.

Recovering from the above error can be done in a variety of ways. The simplest recovery plan is to switch arrangeDelivery for another service, like arrangeShipment (similar to the static recovery case). Another option is to switch to plan dp_2 (see Section 8.2, which is a permutation of dp_1. Since payment occurs before arranging delivery in dp_2, our user can continue using the arrangeDelivery service. If the user is not satisfied with these recovery options, PWWM also offers *replanning* (see Section 8.3), aimed to suggest new plans while taking into account compensation for tasks already carried out.

10 Discussion and Challenges

The Personal Web Workflow vision is just what it is – a vision. We are yet to implement it and experiment with its effectiveness, even though we believe that such an implementation is possible with existing web technologies. This experience would also explicate cases where our framework assumptions, described in Section 5 are too strong. Ideally, they can be addressed using some of the techniques offered in this book.

Regardless of the technological challenges, we believe that the ultimate success of Personal Web Workflow vision described in this paper critically depends on successfully solving three major research problems: (a) effective elicitation of goals and a

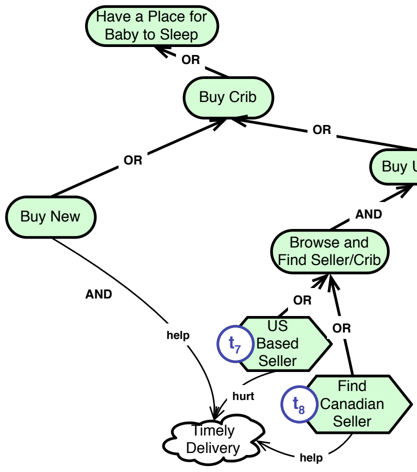


Fig. 20: Model slice relevant to the ”Timely Delivery” soft goal.

variety of other properties that can be used to help produce usable plans and monitor for their successful execution; (b) scalability of the various analyses performed “behind the scenes” in this framework, from relationship identification to planning to monitoring and recovery; (c) creating provisions for collaboration of multiple users in order to accomplish a particular goal (e.g., the crib can be bought not only by the user but by her parents who reside in the US, and now the framework needs to ensure not only that the crib arrives on time, but also that two cribs are not bought accidentally). In what follows, we discuss several proposals related to these issues.

Goal and Preference Elicitation. In this paper, we proposed to rely on non-intelligent keyword search to help turn user narrative into goal models. Although simple and useful, this approach may lead to plenty of false positives which need to be filtered manually by the user. This can be ameliorated by using templates that place restrictions on how natural language can be used. In particular, the user may be asked to use a wizard-like environment that permits only certain sentence patterns. Such patterns have been widely studied in behavioural verification for temporal properties [22]. While the general ideas apply to goal models as well, further research is required to identify, classify and evaluate these patterns.

Web 2.0 provides various opportunities to increase the level of sophistication in goal search For example. if the user owns a blog, the starting point for keyword search could be this blog. Moreover, goal search can take advantage of previous queries that the user placed on the search engine. Some interesting work on this topic has already been done by M. Strohmaier and his colleagues [41, 79].

We also need to experiment further with techniques for elicitation, capture, review and maintenance of user preferences, since these are paramount for the success of our proposed methodology.

Multiple Users: Refinement Step. Recall that our merge example in Section 7 required integration of only two models. In practice, the integration step may involve more models, for example, when the user uses multiple crowd-sourced models to operationalize her desired scenario, or when the notion of "user" represents a group of people rather than just one individual. In this case, the "viewpoints" [70] of the different group members and any pre-existing models used by them need to be combined together, before a desired scenario can be operationalized. This in turn requires the specification of a *collection* of inter-related models. A particularly interesting abstraction that can be used for describing such a collection is that of an *interconnection diagram* adapted from category theory [7]. Interconnection diagrams allow the merge operation to be defined over an arbitrarily large number of models and relationships rather than just pairs of models related by a single relationship. In our previous work, we have already described a model merging operator that works over interconnection diagrams [68].

Scalability: Refinement Step. When multiple models are involved, the merge outcome may become too large and thus too complex for the end-user to comprehend. To address scalability, the modeling environment where goal models are constructed and manipulated needs to provide mechanisms for *slicing* of goal models [43]. The purpose of slicing is to extract the fragments of a model that are relevant to a particular task. For example, the personal web user in our example may be interested in only viewing the goals and tasks that help or hurt the satisfaction of the "Timely Delivery" soft goal, in which case the slice should include three core elements: the "Buy New" goal as well as the two leaf tasks t_7 and t_8 from Figure 11. In addition, since non-root goals and tasks cannot be understood outside their context, the slice should further include the higher level goals and tasks related to the core elements. The slicing process should thus yield the model shown in Figure 20. This slice helps the user narrow down her investigation to a small fragment of the overall model, thus reducing cognitive load and improving comprehension. Leica [43] provides a detailed treatment of slicing for goal models, enabling users to extract slices for various types of reviewing activities often performed on goal models.

Scalability: Planning step. Planning techniques have advanced significantly over the past years, allowing for efficient reasoning about real problems, even despite the complexity of the underlying computational problem [14]. The planner infrastructure we are considering [75] actually benefits from the presence of explicit preferences and a recomposition structure given by the goal model. In its current state, its performance is practical for models involving tens of goal and task elements [46]. Of course, the planning step can become more efficient if we attempt to minimize goal interactions or remove unnecessary non-determinism [8].

Also, whenever we believe that static planning takes too long, we can always switch to dynamic planning, trading off guarantees of satisfaction of pre- and post-conditions and vendor preferences for planner efficiency.

Finally, a replanning step seems like a natural candidate for application of *incremental planning techniques* [31,53,87] which take into account the existing state of the plan and look at best ways of continuing it to achieve the (possibly augmented) goal under (possibly augmented) preferences.

Multiple users: Execution and recovery. When multiple users collaborate to achieve a common goal, properties of interest involve the state of all of their workflows, since we want to check properties such as “exactly one crib should be bought”. Since each user has a local view of the collaboration, it is not clear who should specify such properties. Another issue is checking them since monitors must now “talk” to one another, i.e., include events from all workflows. Techniques for turning a centralized monitor into a set of distributed ones, running in different process servers, have been investigated by the DESERT project [37]. We believe that these results can be used to distribute monitoring in the collaboration scenario.

Scalability: Execution and recovery. Currently, our framework permits the definition of properties that depend only on the order and occurrence of system events. By monitoring the actual *data* exchanged by conversation participants, we could check richer properties that depend on such data, e.g., ensuring that the merchant charged the user credit card exactly the cost of the crib purchase. Of course, checking such properties is computationally expensive [32]. Another problem is that since PWWM allows the creation of highly customized applications, we cannot use techniques like caching to reduce the monitoring overhead of multiple applications running on the same server. However, we can reduce these times by doing client-side monitoring, as proposed in [19, 33].

11 Related Work

In this section, we look at approaches related to the three techniques we used here to realize our personal web vision: goal modeling and operationalization, model merging and matching, and web service monitoring and recovery.

Goal modeling and operationalization. Goal modeling has been used extensively in the context of early requirements engineering for software design [55, 82] to express stakeholder goals at different levels of abstraction and to show the impact of different software design alternatives on these goals. This includes work on acquiring such variability [45], selecting alternatives based on user skills and preferences [36], using goal-models to reason about software configurations [45] as well as incorporating end-user preferences [47]. A variety of techniques for performing automated reasoning about such goal models have been proposed [29, 35, 71]. Some of these, e.g., [13, 30, 85], use planners – the reasoning framework we adopt in our work [47, 75]. Planner-based approaches have the benefit of distinguishing between preferences and mandatory goals. Researchers have also attempted to connect goals with services in a variety of ways, including using intentional-level services [38, 65], generating service oriented architectures from *i** models [15], or reasoning about service compositions or adaptations thereof using goals [6, 18]. The modeling of how web service orchestrations impact end-user goals is therefore a natural adaptation of this work. Our approach also extends this work in a novel direction by integrating it with model merge and web service monitoring and recovery.

Model matching and merging. A significant body of research has been developed on model merging over the years. In their survey [21], Darke and Shanks identify model merging as one of the core activities in viewpoints-based development [27]. Several papers study model merging in specific domains including database schema design [54, 62], use cases [64], goal models [68], class diagrams [1], state machines [12,56,57,80], graph transformation systems [24], and web services [48]. Model merging has also attracted considerable attention in ontology research for handling ontologies originating from different communities. Kalfoglou and Schorlemmer provide a survey of existing approaches to mapping, aligning, and merging ontologies [39].

Our application of merge in the context of personal web borrows from our previous work on merging goal models and state machines. The main prerequisite for a successful application of merge is a precise statement of the overlaps between the models. To assist with this task, we provide in [16] a classification of the different types of model overlaps and the applicability of these overlap types to different modeling notations.

As we discussed in the example in Section 7.3, model merging often requires model matching, i.e., a model management operator for defining relationships between models, as a prerequisite step. Matching is addressed either explicitly or via various forms of thesauri and naming conventions. Applications of matching in software engineering go beyond model merging. In particular, matching may be employed to facilitate reuse of artifacts [52, 66] or to detect inconsistencies [23, 77]. In addition, matching techniques have been used to identify candidate services to replace a service in use when it becomes unavailable or unsuitable due to a change [49, 89].

Web service monitoring and recovery. Monitoring techniques for web services can be roughly divided into *offline*, e.g., [50,51,81], that analyze system events *after* execution, and *online* [3, 4, 42, 61, 74] that monitor the system as it runs. Offline techniques have access to the entire trace and thus can check more complex properties, but do not allow to perform recovery, since errors are detected after the execution has finished. We use online monitoring here.

The approach we use in designing PWWM adapts our previous work on recovery and planning [46, 72], allowing us to create recovery plans *dynamically*, after analyzing an application path that led to an error. Several works [5, 28] have suggested "self-healing" mechanisms for web-service applications that rely on predefined recovery strategies. We intend to investigate whether existing self-healing techniques can be extended to handle the level of dynamism associated with personalized workflows.

12 Summary

In this paper, we proposed a vision of personalizing user experience on the web by allowing users to create and execute their own workflows. The vision, which we call the Personal Web Workflow Methodology (PWWM), is based on using three sets of technologies developed as part of our prior research: goal modeling and operationalization, model matching and merging, and web service monitoring and recovery. PWWM enables (1) elicitation of user goals and preferences, (2) creation of high-level goal models, (3) use of crowd-sourcing to find and put together suitable refined goal models, (4) creation of plans that best accomplish these goals, (5) turning them into executable BPEL orchestrations, (6) using user-, vendor- and community-defined preferences, policies and constraints for runtime monitoring, and (7) user-controllable recovery and re-planning in case the desired workflow fails. Our approach combines a high degree of automation with ultimate personalization – the user can be very involved with every step of the process, or customize her environment ahead of time so that the framework takes care of choosing the most suitable workflows, or rely heavily on crowd-sourced information. While our methodology centers around *customization*, it does not yet address the issue of *collaboration* – where multiple users perform steps towards achieving a common goal.

The proposed methodology creates a number of challenges, some of which are technological (and likely solvable in a very near future), whereas others likely requiring advanced techniques and new research.

References

1. M. Alanen and I. Porres. "Difference and Union of Models". In *Proc. of UML'03*, pages 2–17, 2003.
2. P. Avesani, C. Bazzanella, A. Perini, and A. Susi. "Facing Scalability Issues in Requirements Prioritization with Machine Learning Techniques". In *Proc. of RE'05*, 2005.
3. L. Baresi, C. Ghezzi, and S. Guinea. "Smart Monitors for Composed Services". In *Proc. of ICSOC'04*, pages 193–202, November 2004.
4. L. Baresi and S. Guinea. "Towards Dynamic Monitoring of WS-BPEL Processes". In *Proc. of ICSOC'05*, pages 269–282, 2005.
5. L. Baresi, S. Guinea, and L. Pasquale. "Self-Healing BPEL Processes with Dynamo and the JBoss Rule Engine". In *Proc. of ESSPE'07*, pages 11–20, 2007.
6. L. Baresi and L. Pasquale. "Live Goals for Adaptive Service Compositions". In *Proc. of SEAMS'10*, pages 114–123, 2010.
7. M. Barr and C. Wells. *Category Theory for Computing Science*. Les Publications CRM Montréal, Montreal, Canada, third edition, 1999.
8. A. Barrett and D. S. Weld. "Characterizing Subgoal Interactions for Planning". In *Proc. of IJCAI'93*, pages 1388–1393, 1993.
9. C. Batini, M. Lenzerini, and S. Navathe. "A Comparative Analysis of Methodologies for Database Schema Integration". *ACM Computing Surveys*, 18(4):323–364, 1986.
10. M. Bienvenu, C. Fritz, and S. McIlraith. "Planning with Qualitative Temporal Preferences". In *Proc. of KR'06*, June 2006.
11. G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh. "A Manifesto for Model Merging". In *Proc. of GaMMa'06, co-located with ICSE'06*, 2006.
12. G. Brunet, M. Chechik, D. Fischbein, N. D'Ippolito, and S. Uchitel. "Weak Alphabet Merging of Partial Behaviour Models". *ACM Transactions on Software Engineering and Methodology*, 21(2), 2011. (To Appear).
13. V. Bryl, F. Massacci, J. Mylopoulos, and N. Zannone. "Designing Security Requirements Models through Planning". In *Proc. of CAiSE'06*, 2006.
14. T. Bylander. "Complexity Results for Planning". In *Proc. of IJCAI'91*, pages 274–279, 1991.
15. C. B. Castro, X. Franch, and H. Astudillo. "From i* Models to Service Oriented Architecture Models". In *Proc. of ACT4SOC'10*, pages 52–63, 2010.
16. M. Chechik, S. Nejati, and M. Sabetzadeh. "A Relationship-Based Approach to Model Integration". *J. Innovations in Systems and Software Engineering*, 2011. (To Appear).
17. M. Chechik, J. Simmonds, S. Ben-David, S. Nejati, M. Sabetzadeh, and R. Salay. "Modeling and Analysis of Personal Web Applications: A Vision". In *Proc. of CASCON'10 Personal Web Wkshp.*, 2010.
18. A. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. "Modeling and Reasoning about Service-Oriented Applications via Goals and Commitments". In *Proc. of CAiSE'10*, 2010.

19. S. R. Choudhary and A. Orso. "Automated Client-Side Monitoring for Web Applications". In *Proc. of WEBTEST'09*, pages 303–306, 2009.
20. A. Dardenne, A. van Lamsweerde, and S. Fickas. "Goal-Directed Requirements Acquisition". *Science of Computer Programming*, 20(1-2):3–50, 1993.
21. P. Darke and G. Shanks. "Stakeholder Viewpoints in Requirements Definition: a Framework for Understanding Viewpoint Development Approaches". *Requirements Eng. J.*, 1(2):88–105, 1996.
22. M. Dwyer, G. Avrunin, and J. Corbett. "Patterns in Property Specifications for Finite-State Verification". In *Proc. of ICSE'99*, pages 411–420, May 1999.
23. A. Egyed and N. Medvidovic. "A Formal Approach to Heterogeneous Software Modeling". In *Proc. of FASE'00*, pages 178–192, 2000.
24. G. Engels, R. Heckel, G. Taenzter, and H. Ehrig. "A Combined Reference Model- and View-Based Approach to System Specification". *J. Soft. Eng. and Knowl. Eng.*, 7(4):457–477, 1997.
25. ESSI WSML working group. Web Services Modeling Language (WSML). `http://www.wsmo.org/wsml/`, Accessed August 2011.
26. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, 1998.
27. A. Finkelstein, J. Kramer, B. Nuseibeh, and M. Goedicke. "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development". *J. Soft. Eng. and Knowl. Eng.*, 2(1):31–58, 1992.
28. M. G. Fugini and E. Mussi. "Recovery of Faulty Web Applications through Service Discovery". In *Proc. of SMR-VLDB'06*, pages 67–80, 2006.
29. A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. "Specifying and Analyzing Early Requirements in Tropos". *J. Requirements Eng.*, 9(2):132–150, 2004.
30. G. Gans, M. Jarke, G. Lakemeyer, and T. Vits. "SNet: A Modeling and Simulation Environment for Agent Networks Based on i* and ConGolog". In *Proc. of CAiSE'02*, 2002.
31. G. Giacomo, Y. Lespárance, H. Levesque, and S. Sardina. "IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents". In *Multi-Agent Programming*, pages 31–72. Springer, 2009.
32. S. Hallé and R. Villemaire. "Runtime Monitoring of Message-Based Workflows with Data". In *Proc. of ECOC'08*, pages 63–72, 2008.
33. S. Hallé and R. Villemaire. "Browser-Based Enforcement of Interface Contracts in Web Applications with BeepBeep". In *Proc. of CAV'09*, pages 648–653, 2009.
34. J. Hoffmann, P. Bertoli, M. Helmert, and M. Pistore. "Message-Based Web Service Composition, Integrity Constraints, and Planning under Uncertainty: A New Connection". *J. Artif. Intell. Res. (JAIR)*, 35:49–117, 2009.
35. J. Horkoff and E. Yu. "Analyzing Goal Models – Different Approaches and How to Choose Among Them". In *Proc. of SAC'11*, 2011.
36. B. Hui, S. Liaskos, and J. Mylopoulos. "Requirements Analysis for Customizable Software: A Goals-Skills-Preferences Framework". In *Proc. of RE'03*, pages 117–126, 2003.
37. P. Inverardi, L. Mostarda, M. Tivoli, and M. Autili. "Synthesis of Correct and Distributed Adaptors for Component-Based Systems: an Automatic Approach". In *Proc. of ASE'05*, pages 405–409, 2005.
38. R. S. Kaabi, C. Souveyet, and C. Rolland. "Eliciting Service Composition in a Goal Driven Manner". In *Proc. of ICSOC'04*, pages 308–315, 2004.
39. Y. Kalfoglou and M. Schorlemmer. "Ontology Mapping: The State of the Art". In *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminars, 2005.
40. J. Karlsson and K. Ryan. "A Cost-Value Approach for Prioritizing Requirements". *IEEE Software*, 14(5):67–74, 1997.

41. M. Kroll and M. Strohmaier. “Analyzing Human Intentions in Natural Language Text”. In *Proc. of K-CAP’09*, pages 197–198, 2009.
42. A. Lazovik, M. Aiello, and M. P. Papazoglou. “Associating Assertions with Business Processes and Monitoring Their Execution”. In *Proc. of ICSOC’04*, pages 94–104, 2004.
43. M. Leica. “Scalability Concepts for i^* Modeling and Analysis”. Master’s thesis, University of Toronto, 2005.
44. S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. Easterbrook. “Configuring Common Personal Software: a Requirements-Driven Approach”. In *Proc. of RE’05*, pages 9–18, 2005.
45. S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. “On Goal-based Variability Acquisition and Analysis”. In *Proc. of RE’06*, pages 76–85, 2006.
46. S. Liaskos, S. McIlraith, S. Sohrabi, and J. Mylopoulos. “Representing and Reasoning about Preferences in Requirements Engineering”. *Requirements Eng. J*, 16:227–249, 2011.
47. S. Liaskos, S.A. McIlraith, and J. Mylopoulos. “Integrating Preferences into Goal Models for Requirements Engineering”. In *Proc. of RE’10*, pages 135–144, 2010.
48. N. Liu, J. C. Grundy, and J. G. Hosking. “A Visual Language and Environment for Composing Web Services”. In *Proc. of ASE’00*, pages 321–324, 2005.
49. N. Lohmann. “Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance”. In *Proc. of BPM’08*, pages 132–147, 2008.
50. K. Mahbub and G. Spanoudakis. “A Framework for Requirements Monitoring of Service Based Systems”. In *Proc. of ICSOC’04*, pages 84–93, 2004.
51. K. Mahbub and G. Spanoudakis. “Run-time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience”. In *Proc. of ICWS’05*, pages 257–265, July 2005.
52. N. Maiden and A. Sutcliffe. “Exploiting Reusable Specifications Through Analogy”. *Communications of the ACM*, 35(4):55–64, 1992.
53. E. Marzal, E. Onaindia, and L. Sebastia. “An Incremental Temporal Partial-Order Planner”. In *Proc. of AIPS’02 Wksp. on Planning for Temporal Domains*, pages 26–32, 2002.
54. S. Melnik, E. Rahm, and P. Bernstein. “Rondo: a Programming Platform for Generic Model Management”. In *Proc. of SIGMOD’03*, pages 193–204, 2003.
55. J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu. “Exploring Alternatives During Requirements Analysis”. *IEEE Software*, 18(1):92–96, 2001.
56. S. Nejati and M. Chechik. “Let’s Agree to Disagree”. In *Proc. of ASE’05*, pages 287–290, 2005.
57. S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. “Matching and Merging of Statechart Specifications”. In *Proc. of ICSE’07*, pages 54–64, 2007.
58. OASIS. Universal Description Discovery and Integration Version 2.04. `http://uddi.org/pubs/ProgrammersAPI_v2.htm`, Accessed August 2011.
59. OASIS. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, Accessed August 2011.
60. OASIS. Web Services Dynamic Discovery Version 1.1. `http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.%html`, Accessed August 2011.
61. M. Pistore and P. Traverso. “Assumption-Based Composition and Monitoring of Web Services”. In L. Baresi and E. Di Nitto, editors, *Test and Analysis of Web Services*, pages 307–335. Springer, 2007.
62. R. Pottinger and P. Bernstein. “Merging Models Based on Given Correspondences”. In *Proc. of VLDB’03*, pages 862–873, 2003.
63. J. Rao and X. Su. “A Survey of Automated Web Service Composition Methods”. In *Proc. of SWSWPC’04*, pages 43–54, 2004.

64. D. Richards. “Merging Individual Conceptual Models of Requirements”. *Requirements Eng. J.*, 8(4):195–205, 2003.
65. C. Rolland, R. S. Kaabi, and N. Kraiem. “On ISOA: Intentional Services Oriented Architecture”. In *Proc. of CAiSE’07*, pages 158–172, 2007.
66. K. Ryan and B. Mathews. “Matching Conceptual Graphs as an Aid to Requirements Re-use”. In *Proc. of RE’93*, pages 112–120, 1993.
67. R. W. Saaty. “Decision Making with the Analytic Hierarchy Process”. *Int. J. of Services Sciences*, 1(1):83 – 98, 2008.
68. M. Sabetzadeh and S. Easterbrook. “View Merging in the Presence of Incompleteness and Inconsistency”. *Requirements Eng. J.*, 11(3):174–193, 2006.
69. M. Sabetzadeh and S.M. Easterbrook. “Analysis of Inconsistency in Graph-Based Viewpoints: A Category-Theoretic Approach”. In *Proc. of ASE’03*, pages 12–21, October 2003.
70. M. Sabetzadeh, A. Finkelstein, and M. Goedicke. “Viewpoints”. In P. Laplante, editor, *Encyclopedia of Software Engineering*, pages 1318–1329. Taylor & Francis, 2010.
71. R. Sebastiani, P. Giorgini, and J. Mylopoulos. “Simple and Minimum-cost Satisfiability for Goal Models”. In *Proc. of CAiSE’04*, pages 20–35, 2004.
72. J. Simmonds, S. Ben-David, and M. Chechik. “Guided Recovery for Web Service Applications”. In *Proc. of FSE’10*, pages 247–256, 2010.
73. J. Simmonds, S. Ben-David, and M. Chechik. “Monitoring and Recovery of Web Service Applications”. In *The Smart Internet 2010*, pages 250–288. Springer, 2010.
74. J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O’Farrell, E. Litani, and J. Waterhouse. “Runtime Monitoring of Web Service Conversations”. *IEEE Tran. on Service Computing*, 2(3):223–244, 2009.
75. S. Sohrabi, J. Baier, and S. McIlraith. “HTN Planning with Preferences”. In *Proc. of IJCAI’09*, pages 1790–1797, 2009.
76. S. Sohrabi and S. McIlraith. “Preference-Based Web Service Composition: A Middle Ground between Execution and Search”. In *Proc. of ISWC’10*, pages 713–729, 2010.
77. G. Spanoudakis and A. Finkelstein. “Reconciling Requirements: A Method for Managing Interference, Inconsistency and Conflict”. *Annals of Software Engineering*, 3:433–457, 1997.
78. B. Srivastava and J. Koehler. “Web Service Composition - Current Solutions and Open Problems”. In *Proc. of ICAPS’03*, pages 28–35, 2003.
79. M. Strohmaier, P. Prettenhofer, and M. Kroll. “Explicit User Goals from Search Query Logs”. In *Proc. Web Intelligence/IAT Workshops’08*, pages 602–605, 2008.
80. S. Uchitel and M. Chechik. “Merging Partial Behavioural Models”. In *Proc. of FSE’04*, pages 43–52, 2004.
81. W. M. P. van der Aalst and M. Pesic. “Specifying and Monitoring Service Flows: Making Web Services Process-Aware”. In L. Baresi and E. Di Nitto, editors, *Test and Analysis of Web Services*, pages 11–55. Springer, 2007.
82. A. Van Lamsweerde. “Goal-Oriented Requirements Engineering: A Guided Tour”. In *Proc. of (RE’01)*, 2001.
83. A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
84. W3C. Web Services Description Language (WSDL). `http://www.w3.org/TR/wsdl/`, Accessed August 2011.
85. X. Wang and Y. Lesperance. “Agent-Oriented Requirements Engineering using ConGolog and i*”. In *Proc. Of AOIS’01*, 2001.
86. K. Wiegers. “First Things First: Prioritizing Requirements”. *J. Soft. Development*, 7(9), 1999.
87. B. C. Williams and P. P. Nayak. “A Reactive Planner for a Model-Based Execution”. In *Proc. of IJCAI’97*, 1997.

88. E. Yu. “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering”. In *Proc. of RE’97*, pages 226–235, 1997.
89. A. Zisman, G. Spanoudakis, and J. Dooley. “A Framework for Dynamic Service Discovery”. In *Proc. of ASE’08*, pages 158–167, 2008.