Flexible Composition of Enterprise Web Services

Liangzhao Zeng[1], Boualem Benatallah[1], Hui Lei[3], Anne Ngu[2], David Flaxer[3], and Henry Chang[3]

[1] School of Computer Science and Engineering, University of New South Wales

[2] Southwest Texas State University, Department of Computer Science, San Marcos, Texas

[3] IBM T.J. Watson Research Center, USA

{zlzhao, boualem}@cse.unsw.edu.au, hngu@austin.rr.com, {hlei, flaxer, hychang}@us.ibm.com

Abstract. The process-based composition of Web services is emerging as a promising approach to automate business process within and across organizational boundaries. In this approach, individual Web services are federated into composite Web services whose business logic is expressed as a process model. Business process automation technology such as workflow management systems (WFMSs) can be used to choreograph the component services. However, one of the fundamental assumptions of most WFMSs is that workflow schemas are static and predefined. Such an assumption is impractical for business processes that have an explosive number of options, or dynamic business processes that must be generated and altered on the fly to meet rapid changing business conditions. In this paper, we describe a rule inference framework called DY_{flow}, where end users declaratively define their business objectives or goals and the system dynamically composes Web services to execute business processes.

1 Introduction

Web services technologies are emerging as a powerful vehicle for organizations that need to integrate their applications within and across organizational boundaries. In particular, the process-based composition of Web services is gaining a considerable momentum as an approach for the effective integration of distributed, heterogeneous, and autonomous applications [2]. In this approach, applications are encapsulated as Web services and the logic of their interactions is expressed as a process model. This approach provides an attractive alternative to hand-coding the interactions between applications using general-purpose programming languages [3].

The wide availability and standardization of Web services make it possible to compose basic Web services into *composite services* that provide more sophisticated functionality and create add-on values [5]. For example, a composite service can provide a high-level financial management service that uses individual payroll, tax preparation, and cash management Web services as components. The process model underlying a composite service identifies the functionalities required by the services to be composed (i.e., the *tasks* of the composite service) and their interactions (e.g., control-flow, data-flow, and transactional dependencies). Technologies for modelling and executing business processes are generally referred to as WFMSs.

Our work is motivated by the requirements of dynamic Web services composition. Web environments are highly dynamic, Web services can appear and disappear around the clock. So, approaches that statically compose services are inappropriate. Business process automation technology such as production WFMS can be used to choreograph the component services. However, one of the fundamental assumptions in most production WFMSs [7] [10] is that process schemas are static and predefined. In order to automate business processes, designers need to understand business processes and use modelling tools to chart process schemas. When a particular business process needs to be enacted, a process instance is created from a process schema. In this approach, the designer is required to explicitly define what are the tasks that compose business processes and specify relationships among them. Such an approach assumes that the particular business process is used repeatedly. However, this is impractical to compose services in many application domains. For example, it is extremely difficult, if not impossible, to predefine all composite service schemas for research and development (abbr R&D) processes in the automobile industry since R&D business processes are very dynamic. To meet the constraints and opportunities posed by new technologies, new markets and new laws, business processes must be constantly redefined and adapted. However, this doesn't mean there are no

business rules that govern R&D processes; it doesn't mean that planning for R&D processes is impossible. Indeed, there is a need to have a system that enables automatic generation of process schemas customized to an organization's environment, business policies and business objectives.

At the same time, since the global economy is volatile and dynamic, organizations are changing constantly as well: entering into new markets, introducing new products and restructuring themselves through mergers, acquisitions, alliances and divestitures. Moreover, most composite services are long running, representing complex chains of tasks. Organizations that executing composite services may need to change business processes to adapt the changes in application requirements, technologies, business policies, conditions and constraints. Consequently, runtime modification of composite services is necessary to meet these changes. However, in most process modelling techniques, such runtime modification is a manual procedure, which is time-consuming and costly to enforce. In order to reduce the cost and provide fast responses to these changes, there is a need to automate the runtime modification of composite service schemas.

In this paper, we present the design and implementation of DY_{flow}: a **DY**namic intelligent **flow** for Web services composition. In DY_{flow}, business objectives are declaratively defined, the composite service definitions are generated on demand and can be re-configured at runtime in order to adapt to changes. In a nutshell, the salient features of DY_{flow} are:

- A set of business rule templates for modelling business policies and a rule inference mechanism for generating composite service schemas. We argue that business rules should be independent of individual business processes. They can be re-used to generate different composite service schemas [23]. We propose a rule inference approach to support dynamic service composition. This is different from traditional business process modelling techniques where the business rules are implicitly codified in the process schemas (e.g., data and control flow constraints).
- An adaptive service composition engine that enables runtime modification of composite service schemas. In DY_{flow}, users (e.g., business process managers) can define business rules to modify the composite service schemas at any time. The adaptive service composition engine can automatically incorporate newly added business rules when there is a need to modify composite service schemas at runtime.

The remainder of this paper is organized as follows. In section 2, we introduce a real world example, to be used for further illustration of our approach. This example stems from a product

lifecycle management case study, which provided motivations for our work. Section 3 introduces some basic concepts and definitions in DY_{flow}. Section 4 describes the details on dynamic service composition. Section 5 illustrates the implementation of the DY_{flow} system. Finally, we discuss related work in section 6 and provide a concluding remark in section 7.

2 A Motivating Example

In this section, we present an example of product lifecycle management in the automobile industry. Let us assume that an automobile company decides to build a new prototype of a sedan car. The chief engineer proposes to build the new prototype car by replacing the old `petrol engine` with a new `electric engine`. In order to achieve this business goal (i.e., replacing the engine), a sequence of tasks need to be conducted, which includes: (1) new electric engine development; (2) new parts layout design; (3) parts design, development & manufacturing and (4) assembling and testing (see Figure 1).

1. **New Engine Development**
   The automobile company has two different alternatives to obtain a new electric engine: it can outsource the development of the electric engine to other companies or it can develop the electric engine in-house. If the cost of outsourcing is over $2000K, then the company will design and develop the electric engine in-house. If the company wants to outsource the electric engine, a suitable vendor needs to be selected first. If a domestic vendor is selected, then the national quality control procedure should be applied, otherwise, international quality control procedure needs to be applied. There are many alternative processes in each type of quality control procedure; the selection of processes depends on the requirements of the electric engine, budgets and relationships with vendors, etc.
2. **Parts Layout Design**
   Replacing the engine of a sedan car will require system engineers to conduct parts layout design. This is because the new and old engines may have different 3D specifications. Some old parts may need to be replaced. New unique parts may be required by the electric engine. System engineers will first conduct a 3D layout evaluation to decide whether to have a complete new layout design or to re-use part of the old layout design. The layout feasibility analysis has three possible alternatives: (1) change the electric engine's 3D specification; (2) change related parts' 3D specifications; or (3) change both engine and related part's 3D

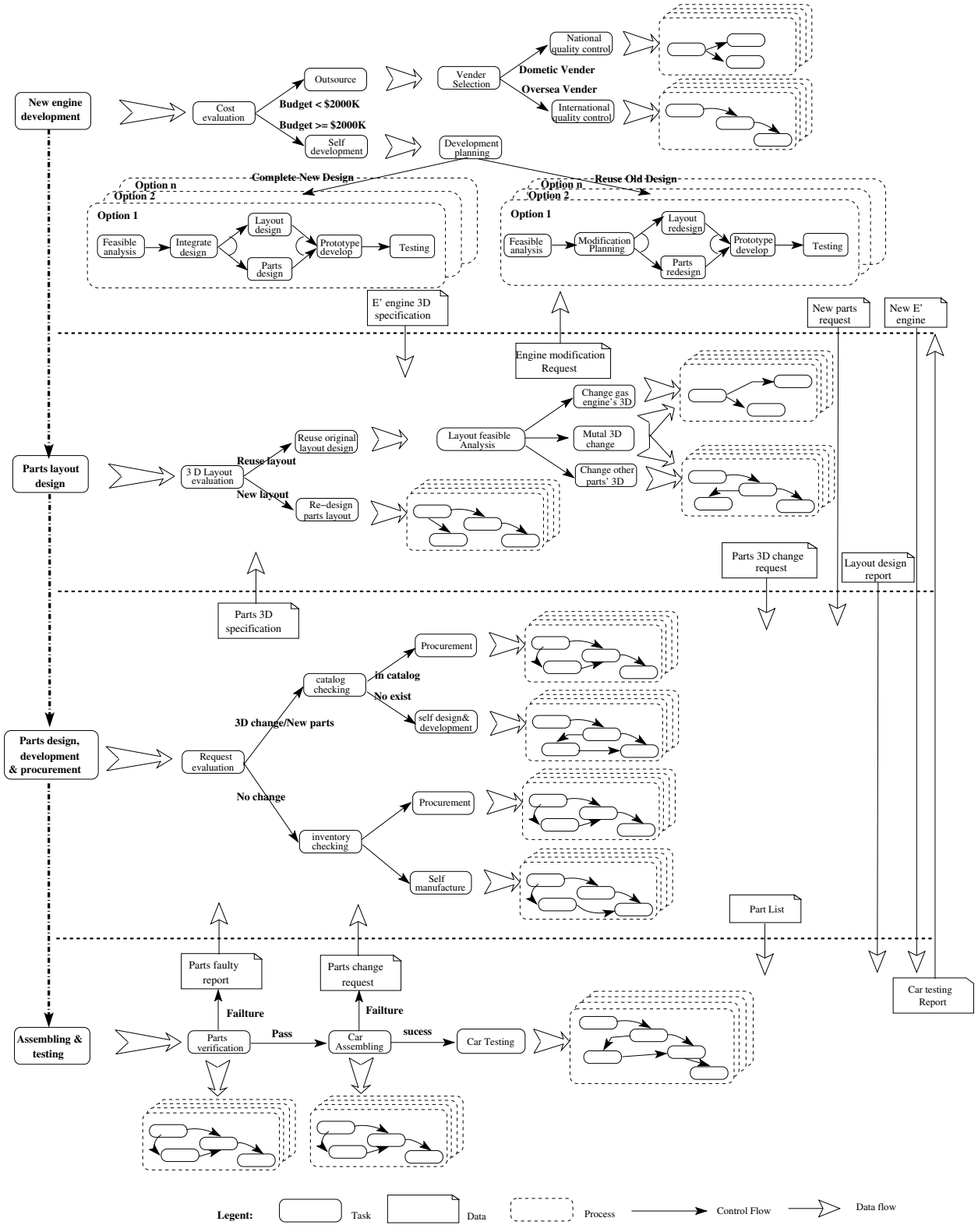

Fig. 1. A Motivating Example

specifications. For each alternative, there is a set of optional processes; the selection of these processes is based on the results of layout feasibility analysis. Inside each process, there are many possible options on which tasks need to be invoked by the process, and lots of possibility on data flows and control flows among the tasks as well.

3. **Parts Design, Development & Procurement**
   Both the new electric engine and the layout design may require the design and the development of new parts for the sedan car. Three basic processes are involved to obtain a new part, namely (1) procurement, (2) self design & development, and (3) self manufacture. The decision on which processes should be adopted is dependent upon the result of request evaluation, catalog checking and inventory checking.
4. **Assembling and Testing**
   When the new electric engine and parts are ready, workshop engineers will verify them first. Any faults on these parts will be reported to the related part obtaining processes. If a part passes the verification, then workshop engineers can assemble the parts. Any problem on assembling will generate a part modification request. When workshop engineers finish the assembling of the car, the test engineers will conduct a sequence of tests, where each test will generate a report. The report may recommend some modifications of the electric engine or other parts. In each step, there are several options for tasks and control flow constraints. The final decisions depend on the type of parts, the budget, testing standards, etc. For example, different countries have different car safety standards. If the automobile company wants to sell the new car in a particular country, then testing based on that country's car safety standard needs to be used.

The process of developing a new sedan car by replacing the `petrol engine` with `electronic engine` will take at least half a year with hundreds of new parts to be designed, developed and manufactured. There are an explosive number of tasks and control flow relationships in this R&D product development process. When pre-defining process schemas, only the general knowledge (i.e., business rules) can be used. The schemes need to consider all the possible options. However, it is almost impossible to predefine process schemas for such a R&D product process since it is too complex and time consuming to exhaustively enumerate all the possible options. This calls for dynamic generation of composite service schemas based on both business rules and context information such as user profiles. For example, during the R&D process, a user (i.e., engineer designer) is assigned to develop a new electronic engine under $2000K budget. In

this case, a composite service can be generated based on current available services, requirement of the new electronic engine, her profile and the budget constraints, without exhaustively enumerating all the possible options in engine development. Another challenge is that this R&D product process is a long running process, during which many changes may occur. For example, during the enactment of this R&D product process, better quality materials or new batteries that can be used for the electric engine may become available. The R&D product process needs to adapt to these changes by modifying process schemes. In production workflows, such adaptations are very costly, especially when workflow schemas are complex. However, if the composite service schema is dynamically generated, adaptation can be done by re-generating composite service schemas. In the following sections, we will use this motivating example to illustrate how the DY_{flow} system composes services to execute business processes.

3 Preliminaries

In this section, we introduce some important concepts and definitions that are used in DY_{flow} system.

3.1 Service Ontology

A service ontology (see Figure 2) specifies a common language agreed by a community (e.g., automobile industry). It defines basic concepts and terminologies that are used by all participants in that community. In particular, a service ontology specifies a *domain* (e.g., `Automobile`, `Healthcare`, `Insurance`), a set of *synonyms*, which is used mainly to facilitate flexible search for the domain (for example the domain `Automobile` may have the synonyms like `Car`). An ontology also specifies *service classes* that are used to define the property of the services. A service class is further specified by its *attributes* and *operations*. For example, the attributes of a service class may include access information such as URL. Each operation is specified by its name and signature (i.e., inputs and outputs). Apart from the service classes that are used to describe functional properties of services, service ontology also specifies *services quality model* that is used to describe non-functional properties of the services, e.g., execution duration of an operation. Services quality model consist of a set of quality dimensions (i.e., criteria). For each quality criteria, there are three basic components: what's the definition of criteria; which service elements (e.g., services or operations) it related to; how to compute or measure

the value of criteria. In DY_{flow}, Web services (elementary services and composite services), business objectives, business rules are specified over defined set of service ontologies.

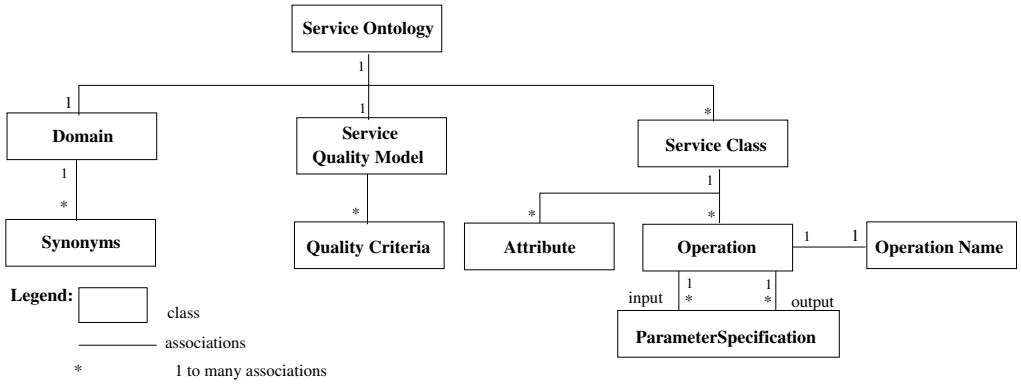

Fig. 2. UML class diagram for service ontology

3.2 Service Description and Advertisement

In order to participate in the DY_{flow} system, services providers need to publish their service descriptions in a service repository. There are two important elements in service descriptions:

- **Service ontology and service class.** A service provider needs to specify which service ontology they use to communicate and which service classes they support. For example, a part testing service provider may specify that it supports service ontology `Cart-Parts` and service class `PartsTesting`.
- **SLAs.** A Service Level Agreement (SLA) is a contract that defines the terms and conditions of service quality that a Web service promises to deliver to requesters. The major component of SLA is QoS information. There are a number of criteria that contribute to a Web service's QoS in SLA. We consider mainly the *execution price* and *execution duration* of each operation. Some service providers publish QoS information in SLAs. Other service providers may not publish their QoS information in their service descriptions for confidential reasons. In this case, service providers need to provide interfaces that only authorized requesters can use to query the QoS information.

3.3 Initial State and Business Objectives

An initial state represents end user's starting point (i.e., initial task) of a business process, while business objectives represent goals (i.e., target tasks) that the user wants to achieve. We

develop an XML schema based language that allows users to specify their initial state and business objectives. In DY_{flow}, both initial state and business objectives are defined in terms of a service ontology. For both initial state and business objective, users need to specify the operation name as defined in service ontology. In initial state, users can provide the constraints on input of the operation; while in the business objectives, users can provide constraints on both input and output of the operation. In Figure 3, the XML document illustrates a business objective where a user wants to change a leaded petrol engine to an unleaded petrol engine for a sedan car. A detailed presentation of the language is out of score of this paper for space reason.

```
< businessObjectives >
  <User Name="Gerg" Role="Chief Engineer" / >
  <targetTask Name="change engine for sedan">
  <ontology-service NAME="Automobile Engine" / >
    < operation NAME="changeEngine" >
      < data-constraint >
        <variable dataName="car" dataItemName="type" />
        <op value="="/>
        <value>"sedan"</value>
      </data-constraint >
      < data-constraint >
        <variable dataName="originalEngine" dataItemName="type" />
        <op value="="/>
        <value>"Leaded Petrol"</value>
      </data-constraint >
      < data-constraint >
        <variable dataName="newEngine" dataItemName="type" />
        <op value="="/>
        <value>"Unleaded Petrol"</value>
      </data-constraint >
    </operation>
  </targetTask >
</ businessObjectives >
```

Fig. 3. An Example of Business Objective

3.4 Business Rule Templates

Business rules are statements about how business is conducted, i.e., the guidelines and restrictions with respect to business processes in an enterprise. We propose using business rule templates to facilitate the description of business policies. There are two categories of business rule templates:

1. **Service Composition Rules.** Service composition rules are used to dynamically compose services. There are three types of service composition rules, namely backward-chain rules, forward-chain rules and data flow rules.

   - *Backward-chain rules* indicate preconditions of executing a task. A precondition can specify data constraints, i.e., some data must be available before the execution of the task. A precondition can also be a flow constraint, i.e., execution of the task requires other tasks to be completed before (i.e., a task must be executed before the task). A backward-chain rule is specified using following structure:

     BACKWARD-CHAIN RULE < `rule-id`>
     TASK < `task`> **CONSTRAINT** <`constraint`>
     PRE-CONDITION <`pre-condition`>

     The following is an example of a backward-chain rule. This rule defines that if a user wants to conduct the `costAnalysis` task on the new part where the part's type is a `car engine`, then the `systemTest` and `clashTest` tasks need to be completed first.

     BACKWARD-CHAIN RULE `bcr1`
     TASK `costAnalysis`
     CONSTRAINT `costAnaysis::partType== 'car engine'`
     PRE-CONDITION `complete_task (systemTest) AND complete_task (clashTest)`

   - *Forward-chain rules* indicate that some tasks may need to be added or dropped as a consequence of executing a given task. Forward-chain rules are defined as ECA (Event-Condition-Action) rules:

     FORWARD-CHAIN RULE <`rule-id`>
     EVENT <`event`> **CONDITION** < `condition`> **ACTION**< `action`>

     In the following forward-chain rule, if task `engineCostAnalysis` is completed and the `makingCost` of the new part is greater than $2000K, then the task `audit_2` needs to be executed after the task `costAnalysis`.

     FORWARD-CHAIN RULE `fcr1`
     EVENT `TaskEvent::complete_task(engineCostAnalysis)`
     CONDITION `(engineCostAnalysis::makingCost> $2000K)`
     ACTION `execute_task(audit_2)`

- *Data Flow Rules* are used to specify data flows among tasks. Each task in a business process may have input and output parameters. For those tasks that require an input, data flow rules can be used to specify the data source. The general form of a data flow rule is given as follows:

  DATA FLOW RULE < `rule-id`>

  CONSTRAINT <`constraint`>

  DATA-SOURCE TASK < `task`> **DATA-ITEM**< `data-item`>

  DATA-TARGET TASK < `task`> **DATA-ITEM**< `data-item`>

  A data source can be a task or human users. For example, in the following data flow rule, task `designEngine`'s output `engineType` provides input for task `testEngine`'s `engineType`.

  DATA FLOW RULE `dfr1`

  DATA-SOURCE TASK `designEngine` **DATA-ITEM** `output::engineType`

  DATA-TARGET TASK `testEngine` **DATA-ITEM** `input::engineType`

2. Service Selection Rules. Service selection rules identify a particular algorithm or strategy to be used for choosing a Web service to execute tasks during the runtime. The general form of a service selection rule is as follows::

   SERVICE SELECTION RULE <`rule-id`>

   TASK< `task`> **CONSTRAINT**< `constraint`>

   SERVICE-SELECTION< `service-selection-method`>

   For each task, there is a set of candidate Web services that can perform the task. Currently, we adopt a Multiple Criteria Decision Making (MCDM) [12] approach to select Web services. However, methodologies other than MCDM such as auction or negotiation can also be adopted by the system to support selection of Web services. MCDM is a configurable decision model for quality-based Web services selection. This decision model takes into consideration some service quality criteria such as execution price, execution duration, etc. Our implementation of MCDM based Web service selection can be found in [22]. In the following service provider selection rule, if the task is `buildEngine` and the engine's weight is greater than 3000Kg, execution price is used as the criteria to select Web services.

   SERVICE SELECTION RULE `ssr1`

   TASK `TaskEvent::start_task(buildEngine)`

CONSTRAINT `(bulidEngine::engine.weight > 3000Kg)`
SERVICE-SELECTION `service_selection(MCDM, execution price)`

In DY_{flow}, we expect domain experts to define the various business rules using the above rule templates. For example, domain experts in service outsourcing may define service selection rules, while domain experts in product life cycle may define service composition rules.

3.5 Organizational Structure

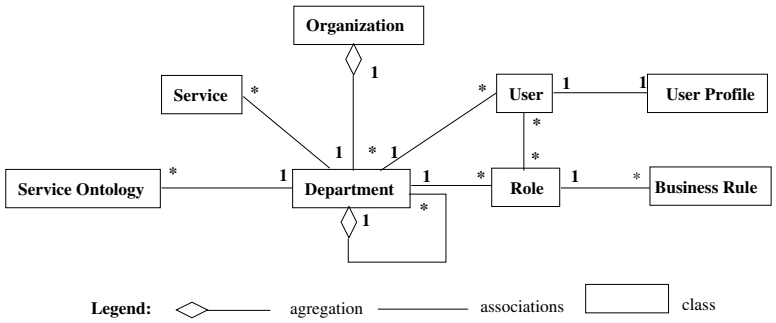

Fig. 4. UML Class Diagram for Organizational Structure

In DY_{flow}, the organizational structure is used to glue the basic elements in an organization (real or virtual) such as *Department*, *Role*, *User*, *Business Rule* and *Service* together(see Figure 4). An organization is subdivided into several departments that again may consist of other departments in a hierarchical structure. Each department is associated with a service ontology, a set of roles, users, and services. Every role is associated with a set of business rules. Every user is assigned one or more roles describing her context-dependent behavior and has her own *user profile*. The information in user profile includes user's personal information, roles in the organization, and preferences. In DY_{flow}, the user profile facilitates the generation and execution of composite services. When the system generates composite service schemas, personal information provides input for executing composite services, user's role information is used to identify relevant business rules, and preferences are used to customize the business rules. For example, in the forward-chain rule `fcr3` (see Table 1), the variable `orderBudgetLimitation` in condition is left uninstantiated. This value will be substituted by a preference (e.g., `orderBudgetLimitation = $20K`) in a user's profile. It should be noted that different users may have different constraints for `orderBudgetLimitation`.

FORWARD-CHAIN RULE `fcr3`
EVENT `TaskEvent::complete_task(partPriceQuatation)`
CONDITION `(partPriceQuatation::price > orderBudgetLimitation)`
ACTION `execute_task(budgetApproval)`

Table 1. A Forward-chain Rule

4 Incremental Service Composition

A composite service schema is defined in terms of service ontologies. It consists of a collection of generic tasks combined in certain ways. In DY_{flow}, we adopt UML statechart [19] to represent the composite services. An example of a composite service schema of the business process for `making new part` is shown graphically in Figure 5. A statechart diagram consists of states and transitions. There are two kinds of states in statechart: actual states and pseudo states. An actual state represents a task that requires invoking a Web service to execute it at runtime. It is labeled by an operation name in a service class of a service ontology. It should be noted that tasks in composite services are not bound to specific Web services at definition time.

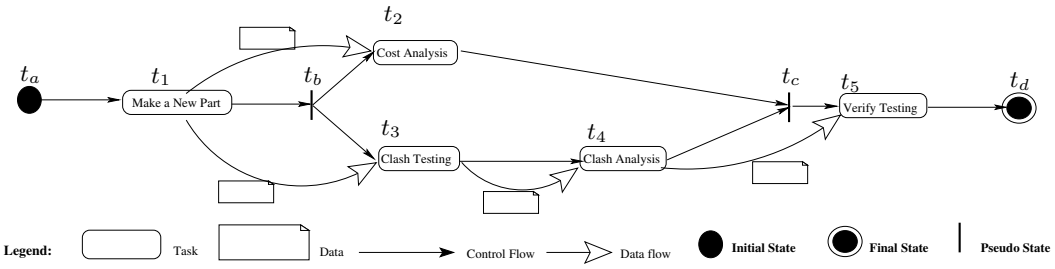

Fig. 5. Defining Composite Service Using a Statechart

Instead of using single composite service schema to represent the whole business process, in DY_{flow}, we identify different levels of composite services and model a business process as a hierarchical structure (i.e., *composition hierarchy*) as shown in Figure 6. A composite service that is used to initiate a new business process is defined as the *top-level composite service*; a composite service that is used to execute a task in a composite service is defined as *task-level composite service*. It should be noted that all the composite services in the composition hierarchy are created on the fly based on user's business objectives using the business rules. The composition hierarchy is built from an initial top-level composite service which is expanded into a hierarchy of composite services at runtime (see Figure 7).

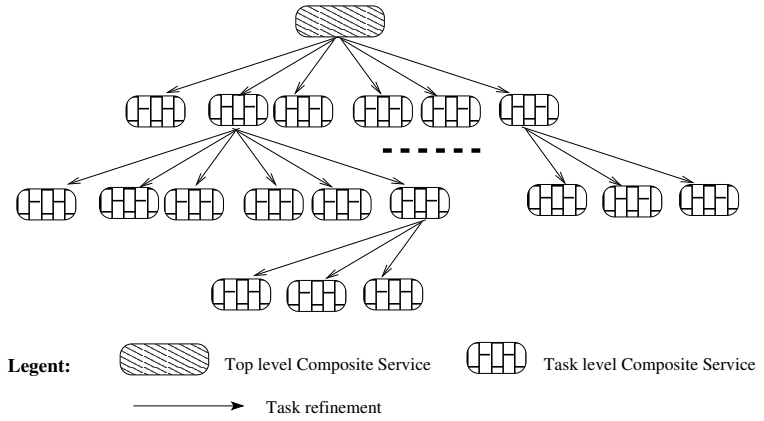

Fig. 6. Composition Hierarchy

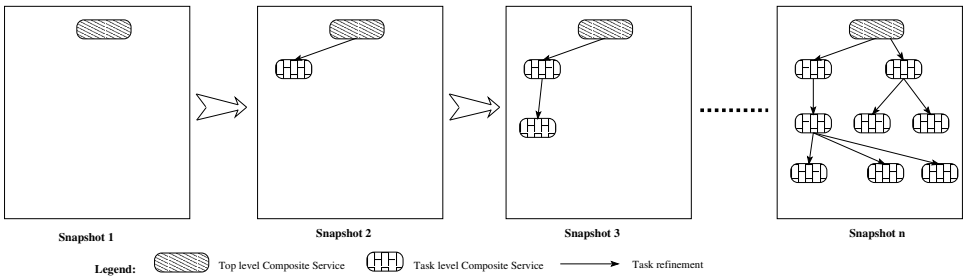

Fig. 7. Snapshots for Composition Hierarchy

Service composition (for both top-level and task-level business process) in DY_{flow} involves three major steps (see Figure 8): composite service generation, selection and execution.

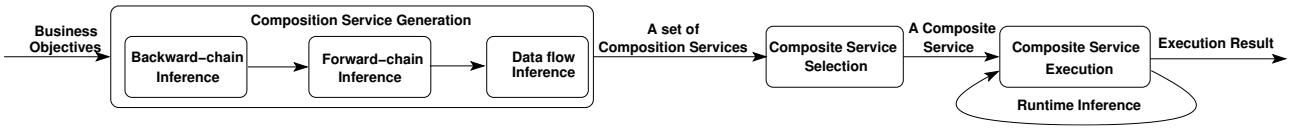

Fig. 8. Composite Service Generation, Selection and Execution in DY_{flow}

1. **Composite service generation.** We formulate the problem of composite service generation as a planning problem, which has three inputs:
   (a) A description of initial state and user's context (e.g., user profile),
   (b) A description of user's business objectives, and
   (c) A set of service composition rules (i.e., domain theory).

   It should be noted, in the input, the service composition rules are associated with the user's role that is specified in user profile. The associations between the business rules and roles are

defined in organizational structure. The output is a set of composite service schemas that can be instantiated and executed to achieve the business objectives. We propose a three-phase rule inference mechanism to generate composite service schemas. During the first phase, the backward-chain inference discovers all the necessary tasks (i.e., backward-chain task) that must be executed to achieve the business objective. The second phase consists of using the forward-chain inference to determine which tasks may potentially be involved as a consequence of executing tasks inferred in previous phase. The final phase involves the data flow inference mechanism. Details about inference algorithms can be found in [23].

2. **Composite service selection.** In some cases, more than one composite service is generated by the three-phase inference mechanism. For example, if there are more than one backward-chain rules for the same task, then there may be multiple ways to archive a business objective. In such case, the choice of delegatee is based on a selection policy involving parameters such as total execution duration, and execution price, etc.

3. **Composite service execution and adaption.** In this step, the system starts executing the selected composite service. At the same time, forward-chain rules are re-applied at runtime to constantly monitor the state of the composite service execution by runtime inference. The runtime inference rules use a broad range of runtime events to drive the rule inference. This differs from the forward-chain inference performed at pre-execution time that assumes all the component services are able to complete the task and use start and termination events only.

5 Implementation Aspects

In this section, we present the current implementation of the DY_{flow} prototype to illustrate the key concepts and ideas in our approach. The prototype (see figure 9) is composed of a *service composition manager*, a *business rule repository*, a *user profile repository* and a *service broker*. All the components of the service composition manager have been implemented in Java and IBM's WSTK [11]. In order to participate in business processes, Web services need to be registered with the service broker. Section 5.1 briefly describes the service broker. Section 5.2 overviews the components of composite manager.

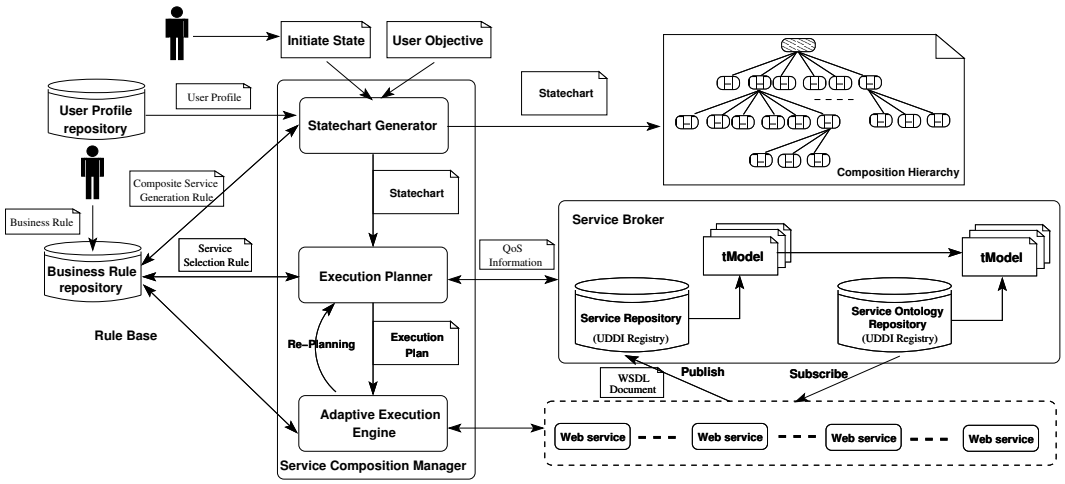

Fig. 9. Architecture of the DY_{flow} prototype

5.1 Service Broker

There are two meta-data repositories in service broker, namely *service ontology repository* and *service repository*. We adopt the UDDI registry to implement both meta-data repositories. We define an XML schema for service ontologies. Each service ontology is represented as an XML document. A separate tModel of type `serviceOntologySpec` is created for each service ontology. The information that makes up a `serviceOntologySpec` tModel is quite simple. There is a tModel key, a name (i.e., service ontology's name), optional description, and a URL that points to the location of service ontology description document.

Based on UDDI API, the service broker provides two kinds of interfaces for both repositories:- the publish interface and the search interface. For the service ontology repository, the publish interface allows an ontology engineer to create new service ontology. It also provides methods to modify the service ontology such as add a new service class, delete an existing service class, etc. The search interface allows service providers and end users to search and browse the existing service ontologies. The search can be based on service ontology's domain name, synonyms, service class, etc. For the Web service repository, the publish interface allows service providers to publish or advertise their service descriptions. While the search interface allows the user to discover Web services by service class name, operation name, input and output data. It should be noted that QoS information is retrieved via generic operations (e.g., getExecutionTime()).

5.2 Service Composition Manager

The service composition manager consists of three modules, namely the *statechart generator*, the *execution planner* and the *adaptive execution engine*.

The statechart generator receives end users' initiate state (i.e., initiate task) and business objectives (i.e., the target task) as input. It locates user's profile in the repository and consults business rules to dynamically generate a set of independent statecharts in XML documents. When a statechart is generated, the execution planner will create an optimal execution plan based on Web services' QoS information in an XML document. Details on execution planning can be found in [22]. The optimal execution plan will be passed to the adaptive execution engine to execute composite services. The execution engine manages the composite service's lifecycle and enables state control, service collaboration and monitoring by executing business rules.

The service composition manager GUI (see Figure 10) provides a single point-of-access to DY_{flow} system. Using the *Business Rule Manager*, users can edit, modify and delete business rules. The GUI also provides a *Profile Manager* that allows users to manage their profiles. The lower panel shows the tool for displaying the diagram of a statechart graphically.

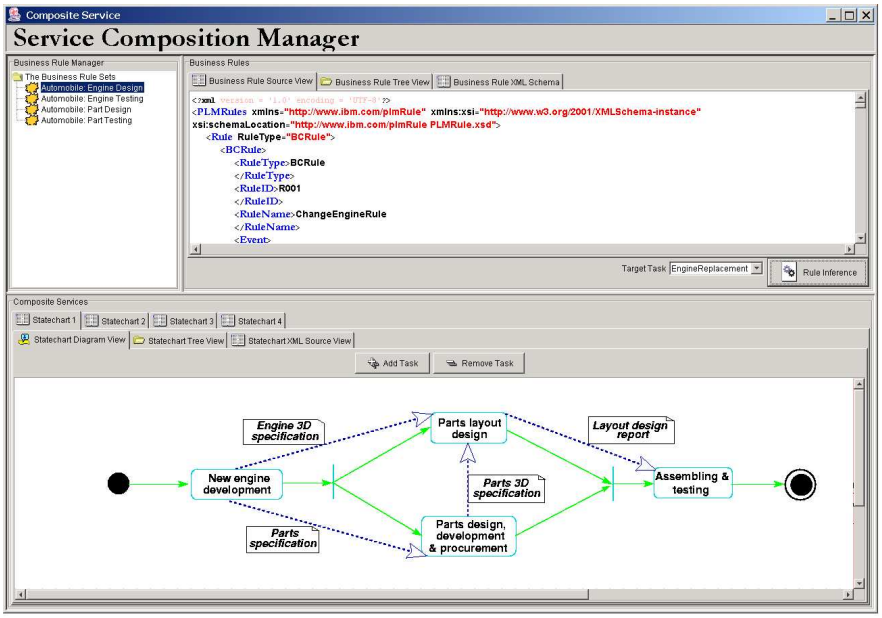

Fig. 10. GUI of Service Composition Manager

5.3 An Application

To illustrate the viability of our approach, we have implemented an automobile R&D application. We used about 100 business rules in this application. The application incrementally generates composite services to manage the `replacing engine` R&D product process (see section 2). The detailed scenario is as follows:-

- Step 1: Creating top level composite service schema
  In this step, the chief engineer provides description of her business objective (i.e., `replacing engine`) as input to the statechart generator. The statechart generator will locate the user's profile and appropriate business rules to generate an XML document that represent a statechart of service. The graphical presentation of the statechart is shown in Figure 10. After creating the top level composite service, the chief engineer will initiate the R&D product process. The task of `new engine development` will be assigned to an *engine designer*.
- Step 2: Creating task level composite service schema
  Assuming that an engine designer is assigned to execute the first task `new engine development` in the top level composite service. Here, `new engine development` is a task in the top level composite service. However, the statechart generator needs to generate a composite service for the engine designer to execute this task. Having the business objective and the engine designer's profile as initial context, the statechart generator can create a task level composite service schema as shown in Figure 11. It should be noted that, for a task in a composite service, either an elementary service is used to execute it, or the statechart generator create a statechart to execute it. For example, for the task of `Cost Evaluation`, since there is no service composition rule for it, an elementary service is used to execute it. However, for the task of `Outsourcing Engine`, since there is a set of service composition rules, the statechart generator create a composite service to execute it.

The above scenario shows that the system only creates the necessary composite service schemas for the R&D product process. It does not enumerate all the possible tasks, control flows and data flows. Instead of using a single large one-level schema to represent the whole R&D product process, we use composition hierarchy that consists of multiple nested composite services to represent the R&D product process. This modular approach allows distinct process to be encapsulated in a composite service. This representation is more scalable and makes it easy to implement runtime modification on composite services.

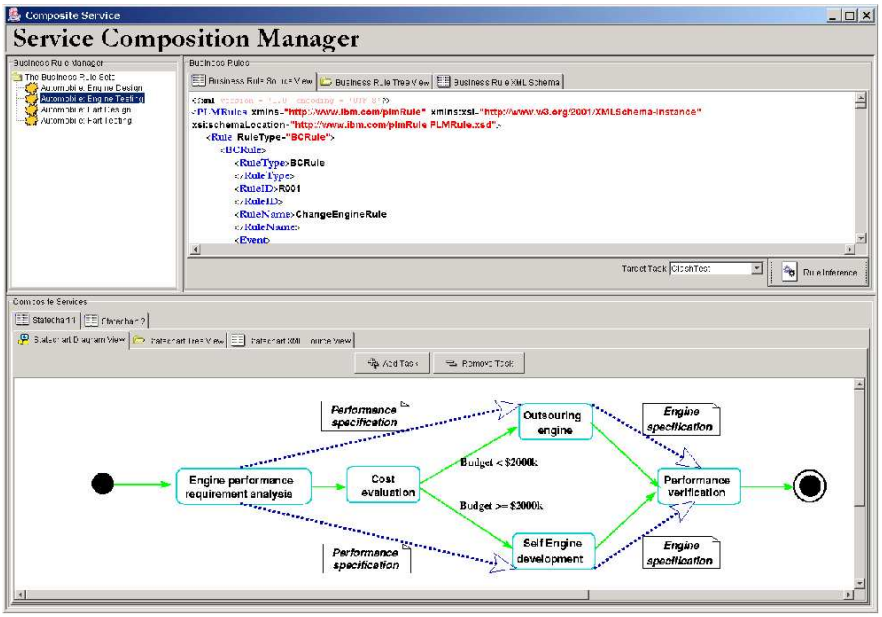

Fig. 11. Task Level Composite Service for New Engine Development

6 Related Work

There are several research efforts in modelling complex process. In this section we review some related work in area of production workflow, Web services standard, and Artificial Intelligent (AI) planning.

Production workflows [7] [14] focus on automating business processes that are characterized by static pattern and high repetition factor. IBM's MQseries workflow [10] use FDL (*flow definition language*) to define workflow schemas. In the WIDE project [6], a workflow management system is built to support distributed workflow execution. ECA rules are used to support exceptions and asynchronous behavior during the execution of distributed workflow instance. In the EvE project [8], ECA rules are used to address the problem of distributed event-based workflow execution, which is a fundamental metaphor for defining and enforcing workflow execution. Typically, production workflows require define workflow schemas in advance, which is not suitable for dynamic Web service composition where the business rules may be modified or created during the execution of business process. However, DY_{flow} uses ECA rules to specify business rules and dynamically generate composite services using those rules at runtime, which can adopt the changes in business rules.

Decision Flow [9] focuses on providing a high level business process specification language with declarative semantics, which are understood by users throughout an enterprise. It provides

an algorithm for eager detection of eligible, needed or necessary tasks to support efficient execution of decision flow. However, a decision flow is predefined and the business rules are hard coded into the decision flow. ISEE [16] introduces events and rules to the business process model. This enables runtime modifications of the business processes. However, all the tasks in the business processes are predefined and the rules cannot be modified dynamically. DY_{flow} composes the business processes on demand immediately before execution and continuously adapts the process as events occur at runtime. Business rules are re-evaluated at runtime to ensure the optimal composite service is used at runtime.

Several standards that aim at providing infrastructure to support Web services composition have recently emerged including SOAP[17], WSDL[21], UDDI[18], and BPEL4WS[4]. SOAP defines an XML messaging protocol for basic service interoperability. WSDL introduces a common grammar for describing services. UDDI provides the infrastructure required to publish and discover services in a systematic way. Together, these specifications allow applications to find each other and collaborate using a loosely coupled, platform-independent protocol. BPEL4WS is the latest attempt to add a layer on top of WSDL for specifying business processes and business interaction protocols. By doing so, it extends the Web services interaction model and enables it to support business transactions. BPEL4WS and DY_{flow} are complementary to each other. The former provides a formalism for defining composite services, while the latter is concerned with how composite services may be derived from business rules. Currently, our prototype uses UML statechart to describe the resultant composite services. The choice of statecharts for specifying composite services is motivated by two main reasons: (i) statecharts have a well-defined semantics; and (ii) they offer the basic flow constructs found in contemporary process modelling languages (i.e., sequence, conditional branching, structured loops, concurrent threads, and inter-thread synchronization). The first characteristic facilitates the application of formal manipulation techniques to statechart models, while the second characteristic ensures that the generated composite services can be adapted to other process modelling languages, like for example, it could easily switch to the BPEL4WS framework.

Early work in AI planning seeks to build control algorithms that enable an agent to synthesize a plan to achieve its goal [20]. Recently, AI planning is used in information gathering and integration over the web [15] [13]. In [1], AI planning is adopted to enable the interaction of Web services, but it requires predefined activity diagrams (i.e., workflow schema) that enumerate

all the possible interactions of Web services. DY_{flow} uses planning algorithm for choosing the most optimal execution plan, but it does not require any predefined workflow schemas.

7 Conclusion

DY_{flow} is a dynamic service composition framework that uses rule inference to support Web service composition. DY_{flow} provides tools for: (i)defining business rules; (ii) generating composite services to execute business processes. Together, these tools provide an infrastructure for a new approach to dynamically compose Web services. Currently, the DY_{flow} platform has been used to generate a business process for an automobile industry. In particular, a relatively complex composition hierarchy that contains 15 composite services (120 tasks) has been created for the `replacing engine` R&D product process. Ongoing research includes optimization of composite services and automatic business rule learning.

Acknowledgments

The work of Boualem Benatallah is partially supported by the Australian Research Council's SPIRT GRANT C00001812.

References

1. M. Aiello, M. Papzoglou, J. Yang, M. Carman, M. Pistore an d L. Serafini, and P. Traverso. A request language for web-services based on planning and constraint sa tisfaction. In *VLDB workshp on Technologies for E-Services (TES)*, LNCS, page 10. Springer, 2002.
2. Boualem Benatallah and Fabio Casati, editors. *Distributed and Parallel Database, Special issue on Web Services*. Springer-Verlag, 2002.
3. Boualem Benatallah, Marlon Dumas, and Zakaria Maamar. Definition and execution of composite web services: The self-serv project. *Bulletin of the Technical Committee on Data Engineering*, 25(4):47–52, 2002.
4. Business process execution language for web services, version 1.0, 2000. http://www-106.ibm.com/developerworks/library/ws-bpel/.
5. Fabio Casati, Ming-Chien Shan, and Dimitrios Georgakopoulos, editors. *VLDB Journal, Special issue on E-Services*. Springer-Verlag, 2001.
6. Stefano Ceri, Paul W. P. J. Grefen, and Gabriel Sanchez. WIDE: A distributed architecture for workflow management. 1997.
7. Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An overview of workflow management: From process modeling to workfl ow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
8. Andreas Geppert and Dimitrios Tombros. Event-based distributed workflow execution with EVE. Technical report, 1998.

9. Richard Hull, Bharat Kumar, Gang Zhou, Francois Llirbat, Guozhu Dong, and Jianwen Su. Optimization techniques for data-intensive decision flows. In *Proceeding of 16th International Conference on Data Engineering*, 2000.
10. IBM MQseries Workflow, 2002. http://www-3.ibm.com/software/ts/mqseries/workflow/.
11. IBM WSTK Toolkit. http://alphaworks.ibm.com/tech/webservicestoolkit.
12. M. Kksalan and S. Zionts, editors. *Multiple Criteria Decision Making in the New Millennium*. Springer-Verlag, 2001.
13. Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Navee n Ashish, Ion Muslea, Andrew Philpot, and Sheila Tejada. The ariadne approach to web-based information integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.
14. Frank Leymann, Dieter Roller, and Andreas Reuter. *Production Workflow: Concepts and Techniques*. Prentice Hall, 1999.
15. Drew Mcdermott. Estimated-regression planning for interactions with web services. In *6th Int. Conf. on AI planning and scheduling*, 2002.
16. J. Meng, S. Y.W. Su, H. Lam, and A. Helal. Achieving dynamic inter-organizational workflow management by integrating business processes, events, and rules. In *Proceedings of the Thirty-Fifth Hawaii International Conference on System Sciences (HICSS-35)*, 2002.
17. Simple Object Access Protocol (SOAP) . http://www.w3.org/TR/SOAP.
18. Universal Description, Discovery and Integration of Business for the Web, 2000. http://www.uddi.org.
19. UML Resource page, 2000. At http://www.omg.org/uml/.
20. Daniel S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.
21. Web Services Description Language (WSDL). http://www.w3.org/wsdl.
22. Liangzhao Zeng, Boualem Benatallah, and Anne Hee Hiong Ngu. Dynamic web service integration and collaboration (submit for publication). Technical report, School of Computer Science and Engineering, University of New South Wales, 2002.
23. Liangzhao Zeng, David Flaxer, Henry Chang, and Jun-Jang Jeng. PLM_{flow}– dynamic business process composition and execution by rule inference. In *VLDB workshp on Technologies for E-Services (TES)*, LNCS. Springer, 2002.