The Synthy Approach for End to End Web Services Composition: Planning with Decoupled Causal and Resource Reasoning[*]

Biplav Srivastava
IBM India Research Laboratory
Block 1, IIT, New Delhi 110016, India
sbiplav@in.ibm.com

Abstract

Web services offer a unique opportunity to simplify application integration by defining common, web-based, platform-neutral, standards for publishing service descriptions to a registry, finding and invoking them – not necessarily by the same parties. Viewing software components as web services, the current solutions to web services composition based on business web services (using WSDL, BPEL, SOAP etc.) or semantic web services (using ontologies, goal-directed reasoning etc.) are both piecemeal and insufficient for building practical applications. *Inspired by the work in AI planning on decoupling causal (planning) and resource reasoning (scheduling), we introduced the first integrated work in composing web services end to end from specification to deployment by synergistically combining the strengths of the current approaches.* The solution is based on a novel two–staged composition approach that addresses the information modeling aspects of web services, provides support for contextual information while composing services, employs efficient decoupling of functional and non-functional requirements, and leads to improved scalability and failure handling. A prototype of the solution has been implemented in the Synthy service composition system and applied to a number of composition scenarios from the telecom domain. The application of planning to web services has also brought new plan and planner usability-driven research issues to the fore for AI.

Introduction

Businesses today need to quickly deliver new applications and adaptively manage them. For example, given the intense competition in the telecom sector, mobile telephony service providers (telcos) need to continually develop compelling applications to attract and retain end-users, with quick time-to-market. On one hand, whenever a competitor introduces a new service, the customer expects a similar or better service within days/weeks. On the other hand, telcos want to target enterprise customers due to their potential for higher margins, with custom-developed value-added services that leverage its telecom and IT infrastructure. Mobile applications often use several, relatively simple building blocks – user profile look-ups, yellow pages, location-tracking services, accounting and billing services, etc. Many of these blocks are already in place but they are not easy to reuse and integrate into new applications because they are not built using standardized frameworks or component models. The telcos inability to use existing components increases their cost and time taken to deliver new services. Moreover, as the applications run, various types of changes might affect their operations. The network configurations can change dynamically, new service providers and business relationships may emerge and existing ones may be modified or terminated. A telco has to address such changes while continually trying to optimize its operations. Currently, any change to the application needs manual intervention and re-composition, which often lags behind the needed business responsiveness.

Service providers need tools and standards–based runtime platforms to quickly develop, deploy and manage interesting applications for their clients. This is where Web services come in as they offer standardized service interface description, discovery and messaging mechanisms. Using them, the process of Web Service Composition and Execution (WSCE) would allow the creation of a service (application) that realizes a new functionality, followed by its deployment and execution on a run–time platform. For industry, this could be the solution to their critical problems in business–to–business or enterprise application integration (Alonso *et al.* 2004; Leymann 2003).

The business world has adopted a distributed systems approach for composition in which the web service instances are described using Web Services Description Language (WSDL), and published to a Universal Description, Discovery and Integration (UDDI) directory, composed into flows in a language like Business Process Execution Language for Web Services (BPEL), and invoked with Simple Object Access Protocol (SOAP). The academia has propounded the AI approach of formally representing web service capabilities in ontologies, and reasoning about their functional composition using goal-oriented inferencing techniques from planning (McIlraith, Son, & Zeng 2001). These approaches by themselves are piecemeal, and insufficient. The former has focused on the execution aspects of composite web services,

[*]We summarize the Synthy line of research on web services composition and adaptation for the AI community. Synthy was possible due to the contributions of Vikas Agarwal, Girish Chafle, Koustuv Dasgupta, Gautam Das, Sugata Ghoshal, Mangala Gowri, Neeran Karnik, Arun Kumar, Ashish Kundu, Anupam Mediratta, Sumit Mittal and Sougata Mukherjea.

without much consideration for requirements capture and the development process. The latter approach has stressed the feasibility of service composition based on semantic descriptions of service capabilities, but its output cannot be directly handed off to a runtime platform for deployment.

We proposed a principled web service composition approach (Agarwal *et al.* 2005a; 2005c) by introducing a differentiation between web service *types* and *instances*. The solution drives the composition process right from specification of the business process, through creation of desired functionality using planning techniques on semantically annotated web services, through generation of a deployable workflow by selection and binding of appropriate service instances that optimize non–functional requirements, to finally deploying and running the composite workflow. The challenges we address are in problem characterization (Agarwal *et al.* 2005b), information modeling (Kumar, Srivastava, & Mittal 2005), handling of functional and non–functional requirements and handling change(Chafle *et al.* 2006).

We begin the discussion by walking through a demonstration scenario using a prototype solution implementation. Then, we give a synopsis of how causal and resource reasoning is decoupled in planning and lay the basis for a scalable planning solution. Next, we describe the end-to-end web service composition problem and our solution based on the idea of decoupling reasoning about services types and instance. We present the implemented system as well as the generalized model. Finally, we discuss the implications of the solution for both the web services and AI communities.

Demonstration Scenario

Suppose a telco is attempting to automate a typical Helpline (or call center) for a washing machine manufacturer. In such an application, a customer calls in to report a problem with her washing machine. This problem needs to be assigned to an agent for resolution. If the problem is such that it could be solved over the phone, a desk-based agent at the call center will be assigned. Otherwise, we need to find an agent in the field who can visit the customer and fix the washing machine. The telco would like to create a set of web services that automate parts of this process to whatever extent possible, and keep aggregating these components to create higher-level composite services. Fig. 1 summarizes the workflow in this Helpline scenario (See (Agarwal *et al.* 2005c) for details.). Here is a sampling of the component services that are available in this scenario: Location tracking, SMS, Call Setup, Agent Expertise data, Problem Classification, Agent Selection. Some of these provide telco-specific functions, e.g., delivering SMS text messages, Location tracking of mobile phones. Others are specific to the application domain, e.g., Problem Classification.

Creating a New Service: For building the Helpline service, the user, who is a developer, may choose to use the tool to explore basic services available, build appropriate composite services, and finally build the Helpline service. Alternatively, the user could ask the tool to build the Helpline service at the outset using the available services, and let the tool search through the set of possible plans. We expect the

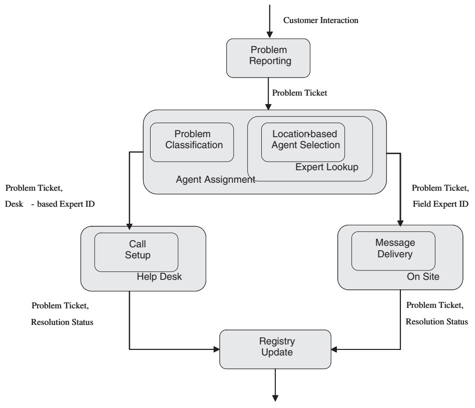
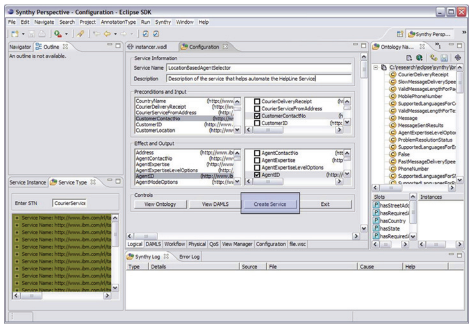

Customer Interaction
Problem Reporting
Problem Ticket
Problem Classification
Location-based Agent Selection
Expert Lookup
Agent Assignment
Problem Ticket, Desk - based Expert ID
Problem Ticket, Field Expert ID
Call Setup
Help Desk
Message Delivery
On Site
Problem Ticket, Resolution Status
Problem Ticket, Resolution Status
Registry Update

Figure 1: Helpline Service


Synthy Perspective - Configuration - Eclipse SDK
File Edit Navigate Search Project AnnotationType Run Synthy Window Help
Navigator Outline
instancer.wsdl Configuration
Ontology Na...
Service Instance Service Type
View Ontology
View DAMLS
Create Service
Exit
Logical DAMLS Workflow Physical QoS View Manager Configuration file.wsc
Synthy Log Error Log
Type Details Source File Cause Help

An outline is not available.

Service Information
Service Name: LocationBasedAgentSelector
Description: Description of the service that helps automate the HelpLine Service

Preconditions and Input
CountryName, CourierDeliveryReceipt, CourierServiceFromAddress, CustomerContactNo, CustomerID, CustomerLocation
☐ CourierDeliveryReceipt
☐ CourierServiceFromAddress
☑ CustomerContactNo
☐ CustomerID

Effect and Output
Address, AgentContactNo, AgentExpertise, AgentExpertiseLevelOptions, AgentID, AgentModeOptions
☐ AgentContactNo
☐ AgentExpertise
☐ AgentExpertiseLevelOptions
☑ AgentID

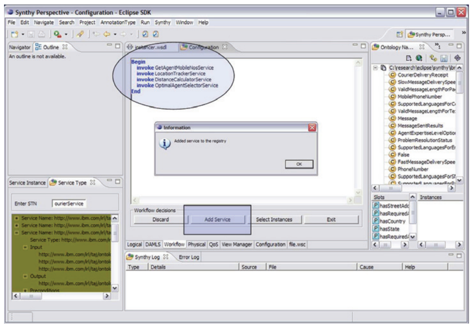
Enter STN: CourierService
Service Name: http://www.ibm.com/irl/ta

Figure 2: Specifying input in the Composition Tool for LAS scenario.

user to prefer the former approach when the scenario is large and the user wants to control the composition.

Suppose the user decides to first build a location-based agent selector service. Given a customer's location and a list of agents out in the field, this service needs to select one of the agents, based on proximity to the customer's residence. This selected agent will then be asked to visit the customer and fix her washing machine. Doing this composition - creating a flow linking together several component services, feeding them the right inputs, etc. - manually takes time and the developer has to know which components exist, and how to connect them up.

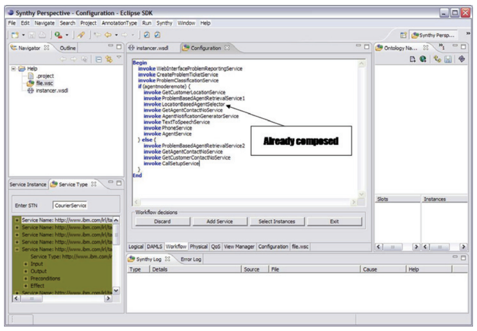
Instead, Synthy discovers the relevant services from amongst the available ones, and creates the control flow between them. The developer only needs to (formally) specify the requirements of the service to be created (see Fig. 2). Fig. 3 shows the plan created for the LAS service. Further, the newly created Location-based Agent Selector (LAS) service itself becomes available as a component (also shown in figure). Each of the services in the plan are web service types whose instances are registered in a service registry. In the physical composition stage, the user is asked to choose a criterion that will guide the selection of instances. For example, the user might choose to select instances having minimum response time, or maximum end-to-end throughput. The user criterion is used to arrive at an optimized phys-



```
Begin
  invoke GetAgentMobileNosService
  invoke LocationTrackerService
  invoke DistanceCalculatorService
  invoke OptimalAgentSelectorService
End
```

Added service to the registry

Figure 3: Logical Plan for the LAS service.



```
Begin
  invoke WebInterfaceProblemReportingService
  invoke CreateProblemTicketService
  invoke ProblemClassificationService
  if (agentmoderemote) {
    invoke GetCustomerLocationService
    invoke ProblemBasedAgentRetrievalService1
    invoke LocationBasedAgentSelector
    invoke GetAgentContactNoService
    invoke AgentNotificationGeneratorService
    invoke TextToSpeechService
    invoke PhoneService
    invoke AgentService
  } else {
    invoke ProblemBasedAgentRetrievalService2
    invoke GetAgentContactNoService
    invoke GetCustomerContactNoService
    invoke CallSetupService
  }
End
```

Figure 4: Logical Plan for Helpline Service.

ical binding for all the service types in the composition and create the executable BPEL workflow. The LAS service can now be reused in creating other flows, such as the one in Fig. 1.

Reusing Pre-composed Service in a New Service: Assume that the previously created composite LAS service has been added to the registry. Now Synthy is invoked for the overall Helpline service (Fig. 1) and the Logical Composer produces the contingent plan shown in Fig. 4. Note that the LAS service is reused. The Physical Composer takes the abstract workflow and generates the appropriate BPEL based on developer's QoS requirements.

Causal and Resource Reasoning in Planning

Given a domain, a set of actions that bring state changes in the domain, an initial state and the desired goal state, the planning problem is to find a sequence of actions (also known as a plan) such that when it is executed from the initial state, a goal state can be reached. Planning is comprised of causal reasoning and resource reasoning. Causal reasoning ensures that for every action in the plan, its preconditions can be satisfied from the effect of another action preceding it within the plan. Causal relationships force sufficient orderings among actions to achieve the goals and furthermore, determine the extent of concurrency possible in a plan. Resource reasoning ensures that all the resources needed for the execution of an action are available for allocation without any resource conflicts. A resource conflict occurs when two actions cannot be assigned the same resource, either due to resource characteristics (e.g., non-sharable resources) or due to domain characteristics (e.g., actions interference). If resources are scarce, the resource allocation may involve freeing and reallocating the limited resource which can add more ordering relationships among actions and effectively serialize the plan.

The RealPlan planning approach(Srivastava 2000a; 2000b; Srivastava, Kambhampati, & Do 2001; Srivastava & Kambhampati 1999) explores the role of resources in a planning domain, and how resource-based and causality-based reasoning can be decoupled in planning so that they are effectively integrated for planning efficiency. A major motivation was to scale planning algorithms to large, realistic domains so that anomalies of existing planners like addition of more resources increasing planning time do not occur. Figure 5 shows how causal and resource reasoning could be decoupled in planning. Given the planning problem, the planner first tries to generate an abstract plan that meets the causal requirements of the plan. Then, the plan is passed on to a scheduler to allocate resources. The resulting plan may have violated some causal requirement and hence, could need re-planning. Similarly, the resource requirements could get violated and re-scheduling be done. RealPlan looks at two possible relationships between the two modules - master-slave and peer-to-peer. In the former, only the causal reasoner can provide feedback to the resource reasoner for plan adjustment while in the latter, either of the modules can give feedback to the other and drive the planning process. Empirical results in a variety of standard planning domains showed that a decoupled strategy leads to better planner scalability and performance.

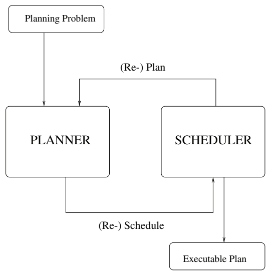
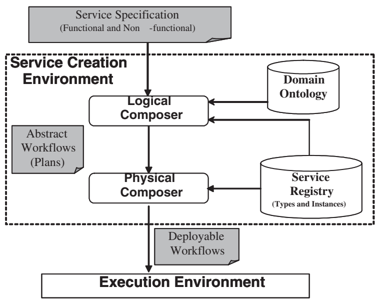


Figure 5: The generalized RealPlan model for decoupling causal (planning) and resource reasoning (scheduling).

The Synthy Approach

Web service composition and execution (WSCE) is the process of realizing the requirements of a new web service (based on specifications) using the existing component web services. Initial approaches for web services composition viewed it as a planning problem where the operations of the available web services are actions, and the goal state is the specification of the composite service(Sirin & Parsia 2004; McDermott 2002). A plan for the planning problem would realize the composite service assuming that the external world during runtime was benign. However, as was later shown, the problem of WSCE cannot be seen as a one-shot



Service Specification
(Functional and Non -functional)
Service Creation Environment
Logical Composer
Domain Ontology
Abstract Workflows (Plans)
Physical Composer
Service Registry (Types and Instances)
Deployable Workflows
Execution Environment

Figure 6: Synthy System

plan synthesis problem defined with explicit goals but rather as a continual process of manipulating complex workflows, which requires solving synthesis, execution, optimization, and maintenance problems as goals get incrementally refined (Srivastava & Koehler 2004).

The specifications for web service composition from an end–user can be decomposed into two parts. The first part deals with the desired functionality of the composite service, called the *functional requirements* (FRs). The second part involves specification of the *non-functional requirements* (NFRs) that relate to issues like performance and availability. The input to an end–to–end web service composition problem is a composite service specification in an appropriate language, e.g., OWL expression for functional requirements, QoS specification for non–functional requirements. We do not restrict the input to any one language, but assume that the specification is done in a manner that would make the subsequent composition feasible. The *end to end service composition problem* can be stated as follows: Given a set of web services and the specifications of a new service, create an executable workflow that stitches together the desired functionality (FRs) from the existing services, while maximizing the satisfaction of the NFRs.

The first novelty we introduced was to differentiate between web service *types*, which are groupings of similar (in terms of functionality) web services, and the actual web service *instances* that can be invoked. We believe that the separate representation of web service type definitions from instance definitions helps in handling different requirements, and different means to optimize them. This, in turn, allows us to work efficiently with large collection of web services. From a RealPlan perspective, we made the distinction between resources and other objects in the planning domain.

The Synthy System: Our composition solution is to use the functional requirements for goal-driven planning to automatically generate abstract plans, and use the non-functional requirements for concretizing the plans by optimally selecting the instances(Agarwal *et al.* 2005a; 2005c). This basic approach and our second novelty, is illustrated in Fig. 6. A **Service Registry** contains information about services available in-house as well as with participating 3rd-party providers. The *capabilities* of each available service *type* are described formally, using domain-specific terminology that is defined in a **Domain Ontology**.

When a new service needs to be created, the developer provides a **Service Specification** to the **Logical Composer** (LC) module. Driven by the specified requirements, the Logical Composer uses generative planning techniques to create a composition of the available service types. Its goal is to explore *qualitatively* different choices and produce an abstract workflow, i.e. a plan (assuming a feasible plan exists) that meets the specified requirements. From a planning perspective, the planner needs to be efficient in this interactive and uncertain domain, the domain could be incompletely modeled, the user has hard and soft constraints, and the number of web service types could be large. We use a contingent planner that uses optional user inputs to efficiently finds plans that matter most to them (Mediratta & Srivastava 2006).

In order to turn the plan into a concrete workflow that can be deployed and executed, specific instances must be chosen for the component services in the plan. The **Physical Composer** (PC) queries the registry for deployed web service instances and uses scheduling and compilation techniques in selecting the optimal web service instances to produce an executable workflow(Agarwal *et al.* 2005a). The focus is now on *quantitatively* exploring the available web service instances for workflow execution.

The workflow generated by the service creation environment must then be deployed onto a runtime infrastructure, and executed in an efficient and scalable manner. The state of the art is to execute the workflow using a workflow engine such as WebSphere Process Choreographer, with data flowing back and forth from this engine to the component web services. Our **Execution Environment** instead orchestrates the workflow in a *decentralized* fashion, with partitions of the flow executing concurrently, in network-proximity with the component services they invoke(Chafle *et al.* 2004). The Synthy system itself consists of the LC and PC stages.

Extended Model: In Figure 7, we give a general model for composition and adaptation. The general model has K abstract workflows from the LC phase and L concrete workflows after the PC phase. There are also ranking functions (R_{AW}, R_{IW}, R_{EW}) to select among multiple alternative workflows and feedback functions (F_R, F_P, F_L) from later stages to give inputs about the operating conditions to the previous stages.

Environmental changes can be handled easily in the extended Synthy model. The key is to increase the number of choices by passing multiple (feasible) compositions across the stages and intelligently select amongst the alternatives available at each stage based on the operating conditions. The details of one possible adaptation approach are described in (Chafle *et al.* 2006).

Relating Synthy to RealPlan: Synthy is motivated by the RealPlan planning approach in doing the composition as phases of causal and resource reasoning. The system can be viewed as implementing the master-slave relationship while the extended model realizes the peer-to-peer relationship.

Impact of Synthy to Web Services and AI

Synthy can be seen as effectively applying the class v/s instance abstraction from Object-Oriented (OO) systems to

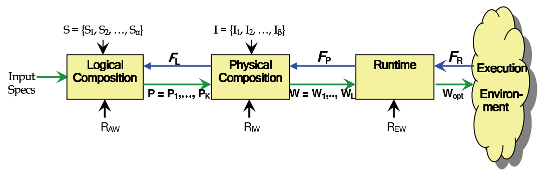


Figure 7: A generalized Synthy architecture for service composition and adapatation.

web services. In existing web services model, all services are individual instances (objects in OO terms) with no grouping relationship amongst them. We introduced the distinction between service classes and service instances, and synergistically used methods that work best at these different levels of abstractions. Planning techniques are used in generating workflow templates based on service models that change infrequently. Multi-objective QoS optimization is used to select instances with more dynamic service instance information. The two contributions - the abstraction of services and the decoupled view of reasoning about them, are increasingly being adopted in web services, grid and general SOA research.

The application of planning to web services has also brought new usability-driven research issues to fore. A few are: (a) Planners have numerous parameters and it is very useful if they could automatically tune their settings to give good performance in the problems of interest (Srivastava & Mediratta 2005). (b) Users preference for plans need to be addressed, like which contingencies are of more interest than others. Also, while a planner is required to generate sound plans, in partially modeled domains, unsound plans can also be useful to drive further modeling. (c) Plans need to be kept around and possibly considered (e.g., reused, dissimilarity measured) in new situations. More attention is needed in metadata and distance functions for handling plans (Srivastava, Vanhatalo, & Koehler 2005). (d) How the planners are built and maintained needs to be aligned with how architecture for large applications are realized.

Conclusion

Synthy is an effective approach for end-to-end composition of web services composition to enable building of new applications. It is inspired by the RealPlan approach and synergistically combines the strengths of goal-oriented reasoning and performance driven instance selection by: (a) Ontology matching and composition at the type level with service matchmaking (b) Composition at the physical level with instance selection (c) Deployment onto a decentralized workflow orchestration infrastructure. The approach has also lead to new avenues of research in web services and AI.

References

Agarwal, V.; Chafle, G.; Dasgupta, K.; Karnik, N.; Kumar, A.; Mittal, S.; and Srivastava, B. 2005a. Synthy: A system for end to end composition of web services. *Journal of Web Semantics, Vol. 3, Issue 4*.

Agarwal, V.; Chafle, G.; Dasgupta, K.; Mittal, S.; and Srivastava, B. 2005b. Evaluating Planning based Approaches for End to End Composition and Execution of Web Services. In *Proc. AAAI-05 Wk. Exploring Planning and Scheduling for Web Services, Grid and AC*.

Agarwal, V.; Dasgupta, K.; Karnik, N.; Kumar, A.; Kundu, A.; Mittal, S.; and Srivastava, B. 2005c. A service creation environment based on end to end composition of web services. In *Proc. 14th WWW, Chiba, Japan*.

Alonso, G.; Casati, F.; Kuno, H.; and Machiraju, V. 2004. Web Services: Concepts, Architecture and Applications. Springer Verlag.

Chafle, G. B.; Chandra, S.; Mann, V.; and Nanda, M. G. 2004. Decentralized Orchestration of Composite Web Services. In *Proc. 13th WWW, NewYork*.

Chafle, G.; Dasgupta, K.; Kumar, A.; Mittal, S.; and Srivastava, B. 2006. Adaptation in Web Services Composition and Execution. In *IBM Research Report*.

Kumar, A.; Srivastava, B.; and Mittal, S. 2005. Information Modeling for End to End Composition of Web Services. In *Proc. 4th ISWC*.

Leymann, F. 2003. Web Services: Distributed Applications Without Limits. In *Proc. of the Intl. Conf. on Business Process Management (BPM)*.

McDermott, D. 2002. Estimated-Regression Planning for Interactions with Web Services. In *Proc. AIPS*.

McIlraith, S.; Son, T. C.; and Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web.* 16(2):46–53.

Mediratta, A., and Srivastava, B. 2006. Applying Planning in Composition of Web Services with a User-Driven Contingent Planner. Technical Report RI06002, IBM.

Sirin, E., and Parsia, B. 2004. Planning for Semantic Web Services. In *Semantic Web Services Workshop at 3rd International Semantic Web Conference*.

Srivastava, B., and Kambhampati, S. 1999. Scaling up Planning by Teasing out Resource Scheduling. In *ECP-99.*

Srivastava, B., and Koehler, J. 2004. Planning with Workflows - An Emerging Paradigm for Web Service Composition. In *ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services*.

Srivastava, B., and Mediratta, A. 2005. Domain-Dependent Parameter Selection of Search-based Algorithms Compatible with User Performance Criteria. In *AAAI, 1386-1391.*

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in RealPlan. *Artificial Intelligence* 131(1-2):73–134.

Srivastava, B.; Vanhatalo, J.; and Koehler, J. 2005. Managing the Life Cycle of Plans. In *IAAI/AAAI, 1569-1575.*

Srivastava, B. 2000a. Effective Planning by Efficient Resource Reasoning. In *Ph.D. Dissertation. Arizona State Univ., USA*.

Srivastava, B. 2000b. RealPlan: Decoupling of Causal and Resource Reasoning in Planning. In *Proc. AAAI.*