RI 04001 09 March 2004 Computer Science

IBM Research Report

A Software Framework for Applying Planning Techniques

Biplav Srivastava
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

A Software Framework for Applying Planning Techniques

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

Keywords: planning architecture, software reuse, applications

Abstract. Enterprise applications are built along software architectures that allow componentization of building blocks, large-scale reuse through design patterns and frameworks, and easy upgradation/maintenance. As more and more applications of planning are emerging in diverse areas that employ many of the planning-related techniques like reachability and relevance analysis, causal reasoning for plan existence, plan synthesis, temporal reasoning and what-if analysis for bounds on optimal plans, the focus is changing from only planner performance to the overall software process of implementing these techniques. Over time, the implementation of the needed planning techniques has to be maintained and upgraded to better alternatives, just like any other software component.

However, the process of implementing planners (and related techniques) is still to start from scratch as there is no common reuse framework. This not only slows the speed of implementation but also inhibits adoption of better techniques in future. In this paper, we present a Java-based programmatic framework called *Planner4J* to build and reuse planning components in business applications. Experience with Planner4J has shown that it is effective for building and maintaining a variety of planners and techniques in Java, which are being used in different applications requiring planning capabilities.

1 Introduction

Today, more and more usage of planning techniques [17] are being found in enterprise applications like data integration[8], web services composition for application middleware [2, 15] marketing campaigns design[18] and system management (e.g., autonomic computing[1]). By planning techniques, we refer to any automated analysis that operates on action and goal specification. Examples are reachability and relevance analysis, causal reasoning for plan existence, plan synthesis, temporal reasoning, what-if analysis for bounds on optimal plans and execution monitoring. Enterprise applications (also called business applications) are built along software architectures that allow componentization of building blocks, large-scale reuse through design patterns and frameworks, and easy upgradation/maintenance[3]. The objective is as much to reduce the total cost of ownership (TCO) of the application as to provide an efficient solution. They demand implemented planning components to not only focus on performance but also to the overall software process of implementing these techniques.

In the past, planning has been applied in specialized domains like space mission control (e.g., NASA's RAX planner[7]) where domain specific details would dictate customized planning and execution approaches. Planning software would be written from scratch in these performance-constrained domains and reuse remained a non-issue. In academic research, the trend continues to be to download publicly available implementations of influential planners like UCPOP[11] and Graphplan[1], learn the details and modify it for the applications. These original systems were not built for reuse but as efficient implementations of appropriate algorithms.

The success of planning in an application depends as much on the planning techniques used as on the way it is embedded into the runtime. The lack of domain independent planning reuse infrastructure makes it hard to understand the role of planning in an application (are the planning needs really special or was the implementation ad hoc?), slows future upgradation to planning advancements and inhibits solution reuse. In this paper, we present a Java-based programmatic framework called *Planner4J* to build and reuse software components for implementing planning techniques in applications. The benefit Planner4J brings on the usability front is to provide the user (developer of planning applications) with a consistent view of planners so that they are not restricted to any one type of algorithm and can easily select the best-in-class, efficient and expressive planner that they may need over time. Note that though the Planning Domain Description Language (PDDL[5]) representation may also seem to give a consistent view of a planner to the user, it only provides the specification at the level of data inputs and outputs, and not a common programmatic access interface. On the software engineering front, the benefit to the user (developer of planning techniques) is that there is a well-tested common infrastructure of components and patterns so that while developing new planners or techniques, much of the existing components (existing code) can be reused.

Planner4J has been used to build a variety of planners implemented in Java, which are being employed in different applications requiring planning capabilities. Specifically, reference implementations of a classical (heuristic search space) planner, a metric planner and an Hierarchical Task Network planner have been developed in Planner4J which reuse much of their underlying components. In addition, various analysis components are available, e.g., for bi-level planning graph construction, relevance and reachability evaluation, that can be used to improve existing planners by getting more accurate heuristic estimations or build new types of planners.

Experience with using Planner4J in applications has shown that it can effective in bringing diverse planning capabilities into different applications by promoting large-scale reuse and easing upgradation/maintenance. It has been incorporated into a publicly available general-purpose agent building environment (ABLE[2]) [14] and

[1] Also see http://www.research.ibm.com/autonomic/.

[2] At http://www.alphaworks.ibm.com/tech/able.

fielded in system management applications. Components of Planner4J have been easily used to build a decision-support tool for Software Project Management[13]. Other enterprise applications under development are a web services composition tool for assembling, deploying and executing composite web services[12], semantic search of web service directories and auto-recovery in systems management.

The paper is organized as follows: we start with considerations in designing a software framework for reuse and then present the Planner4J planning framework. We next discuss the evaluation of Planner4J in building and maintaining planning techniques and application, and conclude with an overview of contributions and future work.

2 Considerations for a Planning Framework

A framework is a reusable design for a specific part or whole of a software consisting of a set of abstract classes, interfaces and the inter-relationship among their instances[3]. It is an object-oriented design but it does not have to be implemented in an object-oriented language. The framework provides a context for the components in the software collection to be reused. The performance of the software component itself is not influenced by whether it is architected in a framework design or not, but such a software is usually faster to develop, easier to understand and maintain, and better used by end applications.

In building the Planner4J planning framework, our objectives are:

1. Improve usability. We want to provide the end application of planning with a consistent view of planners so that they are not restricted to any one type of algorithm and can easily select the best-in-class, efficient and expressive planner that they may need over time.
2. Low footprint. Applications can use only the pieces that they need and the framework should not impose additional dependencies.
3. Improve code reuse. The framework is architected around essential core concepts of planning so that while developing new planning techniques/ planners, much of the existing components can be reused.
4. Extensibility and Scalability. The application should be able to modify and extend the planning capabilities over time in a well understood context.

3 The Planner4J Planning Framework

Planner4J consists of important abstract classes and interfaces that are required to define and solve a planning problem, viz., action, state, problem, domain and plan, and their reference implementations that can be further reused and extended. It is arranged as a layering of modules (see Figure 1). Planner4J-Core is the core module made up of packages containing generic interfaces and implementations of common capabilities like command-line processing and search queue(see Table 1). All Planner4J planners share the Planner4J-Core, thus promoting reuse and extensibility in the Planner4J design.

3.1 Planner4J ClassicalPlanner

The Planner4J-Classical module contains the reference implementation for the Planner4J-Core interfaces (see Table 2). It implements a

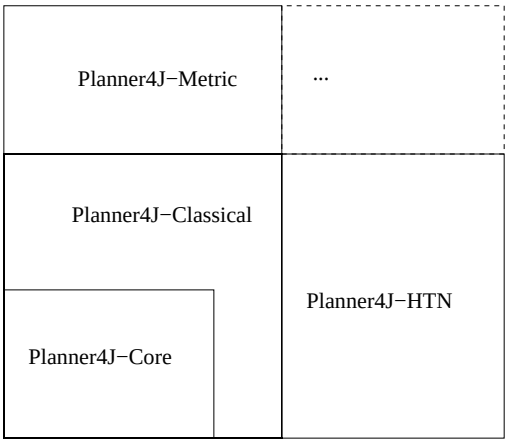

Figure 1. *Layering of modules in Planner4J.*

"classical planner" (STRIPS) that can solve planning problems represented in PDDL1. Each package contains implementations of interfaces defined in the corresponding package of Planner4J-Core module. The module also contains an additional package to house test case driver programs.

Package	Interfaces	Implementations
≺prefix≻.planners	IPlanner IPredicate IDomain IProblem IAction IPlanSolution	Options
≺prefix≻.parsers	IParser	
≺prefix≻.state	IState	
≺prefix≻.search	ISearch IPlanningGraph	SearchQueue

Table 1. Planner4J-Core components. ≺prefix≻ refers to placement of packages in conjunction with the deployment descriptors.

Package	Implementations	Helpers
≺prefix≻. planners	ClassicalPlanner PredicateImpl DomainImpl ProblemImpl ActionImpl PlanSolutionImpl	ActionManager PredicateManager HelperUtil
≺≻.parsers	PDDL1Parser	
≺≻.state	StripsStateImpl	
≺≻.search	StateSpaceSearchImpl BiLevelPlanning- -GraphImpl	PlanningGraph- -HeuristicsCalculator LevelDataItem MutexManager
≺≻.test	TestClassicalPlanner	

Table 2. Planner4J-Classical components.

In the planner package, ClassicalPlanner implements the IPlanner interface and drives the planning process. Additionally, helper classes are defined to manage action and predicate objects. The

StripsStateImpl implementation of IState contains routines to record information about literals and reason with states. The StateSpaceSearchImpl class implements a forward as well as a backward state space search regime guided by a heuristic function. The direction of the search is customizable with a switch provided in the Planner4J-Core Options class. The search package contains support for a Planning Graph[1], which can be used to calculate more accurate heuristics[10] or implement Graphplan within Planner4J. Each planner implementation contains a test program to illustrate how the planner can be programmatically invoked.

Planner4J allows additional planners to be built using the Planner4J-Core and Planner4J-Classical modules. Note that though the use of Planner4J-Classical module is optional, the interfaces in Planner4J-Core have to find some implementation in any planner built within Planner4J and Planner4J-Classical provides ready implementations for components that a planner may not particularly want to customize. Different types of planning (Classical, Metric or HTN[3]) change the representation of the actions and the states, but not the basic search characteristic of the problem. This effect can be modeled as changing the heuristic calculation function that is used to measure distance between states of the underlying planner. At the heart of all the implemented planners is a heuristics-driven state space search algorithm which is tuned by planner-specific heuristics and generic parameters like search direction.

3.2 Planner4J MetricPlanner

Now consider how a new planner may be developed. Also implemented in Planner4J is a restricted metric temporal planner that can reason about cost and performance of actions in generating a feasible plan. It reuses the implementation of Planner4J-Classical as much as possible. In Table 3, the components of the Planner4J-Metric are summarized.

MetricPlanner implements the IPlanner interface. In comparison to PDDL1, actions in this case contain annotations compliant with PDDL2 for cost and duration of actions. As a result, the file parsing routines (MetricParser), the action implementation (MetricActionImpl), the implementation for planning solution (MetricPlanSolutionImpl), and the heuristic evaluation function (MetricStateSpaceSearchImpl) have to be primarily extended/changed. However, even these implementations can reuse and extend the corresponding implementations in Planner4J-Classical.

In the general representation of metric temporal planning, predicates can have duration. We will support this in future and that would require extensions to predicate and state related implementations.

Package	Implementations	Helpers
≺prefix≻. planners	MetricPlanner MetricActionImpl MetricPlanSolutionImpl	HelperUtil
≺≻.parsers	MetricParser	
≺≻.search	MetricStateSpaceSearchImpl	
≺≻.test	TestMetricPlanner	

Table 3. Planner4J-Metric components.

[3] Not an exclusive list. We have identified other types that we want to implement in future.

3.3 Planner4J HTNPlanner

We have implemented a Hierarchical Task Network planner in Planner4J that that can accept descriptions of non-primitive tasks i.e., hierarchical tasks along with the specification how to decompose them, in addition to the PDDL1 primitive tasks. The responsibility of the planner is to produce plans that not only achieve the goals but also respect the specified decompositions. Planner4J-HTNPlanner reuses the implementation of Planner4J-Classical as much as possible. In Table 4, its components are summarized.

The PDDL specification of the domain is extended for non-primitive actions by introducing a *schema* construct that is similar to actions but has an additional *method* field to give expansion (decomposition) of the action[9]. The precondition field of the schema is a place holder to specify necessary conditions for applying the schema which are additional preconditions beyond those of the constituent decomposed actions which should be true to apply the reductions. The effect field records the *primary effects* of the schema for which the merged action should be introduced into the plan. They take care of a basic concern in HTN planning that non-primitive actions should not be used to achieve secondary effects which will unnecessarily produce very complex plans. The method field specifies the choice in selecting a sequence of actions that are to be used for reductions. When there are more than one sequence, a *choice* delimiter is used. The HTNPDDL1Parser implements a parser for the extended PDDL1 domain description while the HTNDomainImpl holds information about the HTN planning domain.

HTNPlanner implements the IPlanner interface. After a schema is parsed, its decompositions are processed to create multiple HTNActionImpl with actions sets recorded in ActionCollection. The heuristic evaluation function (HTNStateSpaceSearchImpl) is extended to let the planner differentiate between actions and prefer non-primitive actions as they result from user provided domain knowledge, rather than primitive actions.

The above discussion was centered on reusing and extending Planner4J-Classical components. In [16], it was shown how planning domain specification at different PDDL levels could be extended slightly to enable a heuristic state space search planner reason with non-primitive tasks and support HTN planning. There, the authors had used and extended the Sapa[4] metric temporal planner. We can similarly extend MetricPlanner too to build a PDDL 3 HTNPlanner quite easily in Planner4J.

Package	Implementations	Helpers
≺prefix≻. planners	HTNPlanner HTNDomainImpl HTNActionImpl ActionCollection Schema	HelperUtil
≺≻.parsers	HTNPDDL1Parser	
≺≻.search	HTNStateSpaceSearchImpl	
≺≻.test	TestHTNPlanner	

Table 4. Planner4J-HTN components.

3.4 Discussion

Planner4J is a collection of abstractions representing essential planning concepts and their reference implementations that can be further reused and extended. We implemented a classical planner, a metric

planner and an HTN planner within Planner4J and discuss the degree of reuse achieved in the next section. Though the discussed planners are based on heuristic search space approach, and can of course be improved with better optimizations/heuristics, *the Planner4J architecture itself is a set of interfaces which does not dictate any particular implementation approach*. We will encourage external planner contributions to extend the spectrum of readily-available planners. One restriction the implemented framework imposes is that the planning components have to be written in Java or there has to be a Java bridge to the implementing language.

4 Evaluating Planner4J

Planner4J has been used to build a rich set of planners techniques which are being applied to various applications detailed in the introduction. Since its main promise is in component reuse and maintenance, we wanted to evaluate how well have we been able to achieve it in developing the reference implementations.

4.1 Experience with using Planner4J

The design and development of Planner4J-Core and Planner4J-ClassicalPlanner was iterative and took 2-3 weeks to stabilize. However, having done that, the first Metric and HTN planners could be implemented in Planner4J in a couple of days. The framework continues to be extended and the latest addition is support for planning graph, which can be used to calculate more accurate heuristics or implement Graphplan within Planner4J.

The biggest advantage of Planner4J, however, has been in the ease of packaging and customizing the right planning components for different applications. While we are continuously upgrading the planning components, we are able to propagate the changes to the differently deployed applications in minutes, using an Integrated Development Editor like Eclipse[4].

4.2 Measuring Reuse

We try to quantitatively understand reuse achieved with Planner4J. Lines of code (LOC) is a standard measure of software complexity in software engineering and we adopt it for our purpose here.

Table 5 shows the lines of code in each package of the Planner4J modules. Since Planner4J-Core consists of both abstractions (i.e., interfaces) and implementations, the latter is also shown separately. We omitted the test package as test programs do not affect the individual planner's functionality.

Package	**Core** L_{core} (L^{ni}_{core})	**Classical** L_{class}	**Metric** L_{metric}	**HTN** L_{htn}
parsers	16	3731	3736	4656
planners	533 (201)	2809	1503	1205
state	288	2407	437	74
search	48 (182)	464		
Total	885 (383)	9411	5676	5935

Table 5. Lines of code (LOC) in various Planner4J packages. For Core, figures in bracket give LOC count for non-interface code.

Using the LOC information, we now calculate statistics about Planner4J under different scenarios(see Table 6). In the first column is the effective LOC for each planner in Planner4J. Since ClassicalPlanner is the basic reference planner, to calculate its effective size, we included Planner4J-Core's LOC with its own LOC. MetricPlanner and HTNPlanner build upon ClassicalPlanner and hence, their effective LOC is the same as LOC in their modules.

In the second column, we consider the scenario if the individual planners had been implemented from scratch. In that case, ClassicalPlanner (and other planners) would not have cared for Planner4J-Core interfaces, as they relate to abstractions, and only implemented the non-interface components (specifically, SearchQueue and Options in Table 1). Therefore, we only consider the latter in the LOC, i.e., $L^{scratch-pess}_{class} = L^{ni}_{core} + L_{class}$. For MetricPlanner and HTNPlanner, we do not know the size of the individual modules if ClassicalPlanner had not been present[5]. But we know that it will be more than the size of ClassicalPlanner because they incorporate more detailed domain modeling and inference reasoning.

The third column calculates how much code has been written *less* in Planner4J compared to if it was written from scratch, as a percentage of the code written from scratch. For example, $L^{payoff-pess}_{class} = ((L^{scratch-pess}_{class} - L^{planner4j}_{class}) / L^{scratch-pess}_{class}) * 100$. The result is a lower estimate because if the size of a planner written from scratch is greater than the lower estimate, its reuse percentage increases. The column shows that for ClassicalPlanner, we end up writing 5% more code to facilitate reuse but for other planners, this translates to a saving of atleast around 40%.

In the fourth and fifth columns, we improve our estimate of payoff due to reuse by being less conservative on the size of individual planners (other than ClassicalPlanner) when implemented from scratch. Since a planner like MetricPlanner, if built from scratch, would have to implement the functionality of ClassicalPlanner in addition to the incremental functionality of metric reasoning, a better estimate for it is the sum of ClassicalPlanner's size from scratch and the incremental size from Planner4J. E.g., $(L^{scratch}_{metric}) = L^{scratch-opt}_{class} + L^{planner4j}_{metric}$.

The fifth column calculates the reuse payoff percentage using the revised estimate of building planners from scratch. For example, $L^{payoff-opt}_{metric} = ((L^{scratch-opt}_{metric} - L^{planner4j}_{metric}) / L^{scratch-opt}_{metric}) * 100$. The column shows that with the same 5% overhead in ClassicalPlanner to facilitate reuse, we may be saving upto about 60% of coding effort.

4.3 Discussion

The evaluation of code reuse is always tricky since it is difficult to select the right measure that captures the functionality of the software components. While LOC is a convenient measure to calculate and generally shows broad trends, it need not be accurate for planning components. However, the above exercise does show that the Planner4J framework can substantially reduce new coding effort. This, along with the initial positive experience in using Planner4J for various planning techniques and applications, leads us to believe that more planning systems should be built around framework principles.

5 Conclusion and Future Work

With more applications of planning emerging, the focus is changing from only planner performance to the overall software process of implementing these techniques. In this paper, we presented a Java-based programmatic framework called *Planner4J* to build and reuse

[4] http://www.eclipse.org

[5] Table 5 only shows the size of the incremental code when ClassicalPlanner is present.

Planner type	**Planner4J** $L^{planner4j}$	**From Scratch (Pess.)** $L^{scratch-pess}$	**Reuse Payoff % (Pess.)** $L^{payoff-pess}$	**From Scratch (Optim.)** $L^{scratch-opt}$	**Reuse Payoff % (Optim.)** $L^{payoff-opt}$
Classical	10296	9794	-5.12	9794	-5.12
Metric	5676	$\geq$ 9794	42.05	15470	63.31
HTN	5935	$\geq$ 9794	39.4	15729	62.27

Table 6. Statistics on payoff due to code reuse based on lines of code (LOC). The reuse payoff ranges from 40-60% based on how one calculates the size of individual planners from scratch.

planning components in applications. We discussed development of three types of planners in Planner4J and evaluated the framework in promoting reuse. Future work will be in expanding the techniques available through Planner4J (e.g., probabilistic reasoning, execution monitoring) and continually improving their performance. We will also encourage external planner contributions to extend the spectrum of readily-available planners.

REFERENCES

[1] A. Blum and M. Furst, 'Fast planning through planning graph analysis', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, ed., C. Mellish, pp. 1636–1642. Morgan Kaufmann, San Francisco, CA, (1995).

[2] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, and G. Mehta, 'The role of planning in grid computing', In Giunchiglia et al. [6].

[3] G. Booch, 'Object-oriented analysis and design with applications (2nd ed)', in *Benjamin-Cummings Publishing Co., Redwood City, CA, USA, ISBN:0-8053-5340-2*, (1993).

[4] Minh B. Do and Subbarao Kambhampati, 'Sapa: A scalable multi-objective heuristic metric temporal planner', in *Journal of Artificial Intelligence Research (JAIR) Special Issue on the Third International Planning Competition*, (2003).

[5] M. Fox and D. Long, *PDDL2.1: An Extension to PDDL for Expressing Temporal Domains*, The AIPS-02 Planning Competition Committee, 2002. Available at http://www.dur.ac.uk/d.p.long/competition.html.

[6] E. Giunchiglia et al., eds. *Proceedings of the 13th International Conference on Artificial Intelligence Planning and Scheduling*. AAAI Press, Menlo Park, 2003.

[7] A. K. Jonsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith, 'Planning in interplanetary space: Theory and practice', in *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, eds., S. Chien, S. Kambhampati, and C. Knoblock, pp. 177–186. AAAI Press, Menlo Park, (2000).

[8] C. Knoblock, S. Minton, J. Ambite, N. Ashish, I. Muslea, P. Philpot, and S. Tejada, 'The ariadne approach to web-based information integration', in *International Journal on Cooperative Information Systems (IJCIS) 10 (1-2) Special Issue on Intelligent Information Agents: Theory and Applications, pp 145-169, 2001.*, (2001).

[9] Drew McDermott, 'The 1998 AI planning systems competition', *AI Magazine*, **21**(2), 35–55, (Summer 2000).

[10] XuanLong Nguyen, Subbarao Kambhampati, and Romeo Sanchez Nigenda, 'Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search', *Artificial Intelligence*, **135**(1-2), 73–123, (2002).

[11] J. Penberthy and D. Weld, 'UCPOP: A sound, complete, partial order planner for ADL', in *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, eds., B. Nebel, W. Swartout, and C. Rich, pp. 103–113. Morgan Kaufmann, San Mateo, (1992).

[12] B. Srivastava, 'Automatic web services composition using planning', in *Proceedings of 3rd Internation Conference on Knowledge-Based Computer Systems*, pp. 467–477, Mumbai, (December 2002).

[13] B. Srivastava, 'A decision-support framework for component reuse and maintenance in software project management', in *IEEE 8th European Conference on Software Maintenance and Reengineering (CSMR 2004), Tampere, Finland.*, (2004).

[14] B. Srivastava, J. Bigus, and D. Schlosnagle, 'Bringing planning to autonomic applications with able', in *To Appear in Proc. IEEE International Conference on Autonomic Computing (ICAC-04), New York, USA.*, (2004).

[15] B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy., 2003.

[16] Biplav Srivastava, 'A Limited Extension of PDDL for Planning with Non-primitive Action', in *Proceedings of ICAPS'03 Workshop on the Competition: Impact, Organization, Evaluation, Benchmarks*, Trento, Italy, (June 2003).

[17] D. Weld, 'Recent trends in planning', *AI Magazine*, **20**(2), (1999).

[18] Q. Yang and H. Cheng, 'Planning for marketing campaigns', In Giunchiglia et al. [6].