Realizing Hybrid Planning Systems as Modern Software Environments

Bernd Schattenberg, Steffen Balzer, Susanne Biundo

We present an architecture for planning and scheduling systems that addresses key requirements of real-world applications in a unique manner. It provides a robust, scalable, and flexible framework through the use of industrial-strength middleware and multi-agent technology. The architectural concepts extend knowledge-based components that dynamically perform and verify the system's configuration; standardized components and communication protocols allow a seamless integration with third-party libraries and application environments. The system is based on a proper formal account of hybrid planning, the integration of HTN and POCL planning. The framework allows to decouple the detection of plan flaws, the computation of plan modifications, and search control. Consequently, planning and scheduling capabilities can be easily combined by orchestrating respective elementary modules and strategies. This platform can implement and evaluate various configurations of planning methods and strategies, without jeopardizing system consistency through interfering module activity.

1 Introduction

Hybrid planning – the combination of Hierarchical Task Network planning with Partial Order Causal Link techniques – turned out to be most appropriate for complex real-world planning applications [8] like crisis management [2, 4]. Here, the solution of planning problems often requires the integration of planning from first principles with the utilization of predefined plans to perform certain complex tasks.

Previous work [18] introduced a formal framework for hybrid planning, in which the plan generation process is properly decomposed into flaw detection and plan modification functions. As an important feature of this approach, an explicit trigger function defines which modifications are suitable candidates for solving which flaws. This allows to completely separate the computation of flaws from the computation of possible plan modifications, and in turn both computations can be separated from search related issues. The system architecture relies on this separation and exploits it in two ways: module invocation and interplay are specified through the trigger function while the explicit reasoning about search can be performed on the basis of flaws and modifications without taking their actual computation into account. This explicit representation of the planning strategy allows for the formal definition of a variety of strategies, and even led to the development of novel so-called *flexible* strategies [18]. The functional decomposition induces a modular and flexible system design, in which arbitrary planning and scheduling system configurations can be integrated seamlessly. A prototype of this architecture served as an experimental environment for the evaluation of flexible strategies against conventional ones as well as a conceptual proof of the expandability of the system with respect to new techniques: the integration of scheduling [16, 17] and probabilistic reasoning [1].

When actually aiming at the employment of this simple architectural nucleus in real-world applications like crisis management, assistance in telemedicine, personal assistance in ubiquitous computing environments, etc., additional features are required, however. Like any other mission critical software in these contexts, also planning and scheduling systems call for highly sophisticated software support. This includes:

1. declarative, automated system configuration and verification – for fast, flexible, and safe system deployment and maintenance, and for an easy application-specific strategic tailoring;
2. scalability, including transparency with respect to system distribution, access mechanisms, concurrency, etc. – for computational power on demand (including load-balancing) without an additional burden for the system developer;
3. standards compliance – for the integration of third-party systems and libraries, and interfaces to other services and software environments.

This paper describes a novel planning and scheduling system architecture which addresses these characteristics. It shows how the formal framework of [18] is implemented by applying modern software technology from service oriented computing and the Semantic Web – in particular middleware and knowledge-based systems – and illustrates how a planning and scheduling system platform for development and evaluation can be created this way.

2 Formal Framework

Our planning system relies on a formal specification of hybrid planning and scheduling [17, 18]: The approach uses a Strips-like representation of action schemata which provide first-order literals for preconditions and effects and indicate the affiliated state transitions as usual by respective add- and delete properties. It distinguishes primitive operators and abstract actions (also called primitive and complex tasks), the latter representing abstractions of *partial plans*. A partial plan consists of complex or primitive tasks, ordering constraints, variable bindings, and causal links to represent the causal structure of the plan. For each complex task, at least one *method* provides a partial plan for implementing it.

Planning problems are defined by an initial partial plan to start from and a domain model – basically a set of primitive

Auszug aus: Künstliche Intelligenz, Heft 1/2007, ISSN 0933-1875, BöttcherIT Verlag, Bremen, **www.kuenstliche-intelligenz.de/order**

and complex task schemata together with a set of methods specifying possible implementations of the complex tasks. A partial plan is a solution to a given problem, if it contains only primitive operators, the ordering constraints and variable bindings are consistent, and the causal links support all operator preconditions without being threatened (i.e., the plan is executable).

Flaws All defects causing a partial plan not to be a solution are made explicit by so-called *flaws*, data structures which describe deficiencies in the plan and in addition allow to classify the problem. A flaw holds a set of references to the affected plan components. The problem, for example, that a plan step s_k interferes with a causal link $\langle s_i, \phi, s_j \rangle$ such that it jeopardizes s_j's executability is represented as a *causal threat flaw* $(\texttt{Threat}, \{\langle s_i, \phi, s_j \rangle, s_k\})$. In the context of hybrid planning, flaw classes also cover the presence of abstract actions in the plan, ordering and variable constraint inconsistencies, unsupported preconditions of actions, etc.

Generating flaws is encapsulated by *detection modules* which are functions that take a partial plan as an argument and return a set of flaws. Without loss of generality we assume that there is exactly one such function for each flaw class. The function for the detection of causal threats in a plan P, for example, is defined as follows: $\text{det}_{\texttt{Threat}}(P) \ni (\texttt{Threat}, \{\langle s_i, \phi, s_j \rangle, s_k\})$ if P's ordering relation allows for s_k being placed between s_i and s_j and the variable bindings of P are consistent with substitutions such that the effects of s_k undo the protected condition ϕ.

Modifications Plan refinement steps are required to obtain a solution out of a problem specification. These steps eliminate flaws and are called *plan modifications*. They explicitly represent changes to the plan structure and consist of sets of elementary additions and deletions of plan components. Adding an ordering constraint between plan steps s_i and s_j, for example, is described as: $(\texttt{AddOrd}, \{\oplus(s_i \prec s_j)\})$. Other examples of hybrid planning modifications are the insertion of new plan steps, the insertion of variable bindings and causal links, and the expansion of complex tasks according to appropriate methods.

As with the flaws, the generation of plan modifications is implemented through *modification modules*. These functions take a plan and a set of flaws as arguments and compute all possible plan modifications that address the flaws. So, "promotion" and "demotion" as an answer to a causal threat means that $\text{mod}_{\texttt{AddOrd}}(P, F)$ contains the modifications $(\texttt{AddOrd}, \{\oplus(s_k \prec s_i)\})$ and $(\texttt{AddOrd}, \{\oplus(s_j \prec s_k)\})$ if F contains a causal threat flaw $(\texttt{Threat}, \{\langle s_i, \phi, s_j \rangle, s_k\})$, for example.

Refinement-based Planning Obviously some classes of modifications address particular classes of flaws while others do not. This relationship is explicitly represented by the so-called *modification trigger function* α which relates flaw classes to suitable modification classes. This includes, for example, that causal threat flaws can *in principle* be solved by expanding abstract actions which are involved in the threat, by promotion or demotion, or by separating variables through in-equality constraints (cf. [2, 18]):

$$\alpha(\text{det}_{\texttt{Threat}}) = \text{mod}_{\texttt{ExpandTask}} \cup \text{mod}_{\texttt{AddOrd}} \cup \text{mod}_{\texttt{AddVarConstr}}$$

Apart from serving as an instruction which modification generators to consign with which flaw, the definition of the trigger function gives us a formal criterion for discarding non-refineable plans: For any sets of detection and modification modules, if any single flaw is not answered by at least one of the α-triggered modification modules, the current plan cannot be refined into a solution.

Based on the presented modules, a generic planning algorithm was developed in [18] which iteratively collects flaws from detection modules, passes them to assigned modification modules, and finally selects plan modifications for execution on the current plan until no more flaws are detected. It also demonstrated how modification selection strategies are formally defined in this framework as *planning strategy modules* and illustrated their potential. Several adaptations of strategies taken from the literature were presented, as well as a set of novel *flexible* planning strategies. The latter exploit the explicit flaw and modification information, which allows for selection schemata that are not defined over flaw or modification type preferences, but perform an opportunistic way of plan generation. In first experiments, a set of flexible and classical, fixed strategies competed on a former planning competition benchmark for hierarchical planning systems. It turned out, that flexible strategies are not only competitive to their fixed ancestors, but also show high optimization potential and can easily be applied to various system configurations.

3 Architecture Overview

As the expositions of the previous sections already suggest, the basic architecture of Panda (*Planning and Acting in a Network Decomposition Architecture*) is that of a multi-agent based blackboard system. The blackboard holds the partial plan and agent societies map directly on the module structure, with the agent metaphor providing maximum flexibility for the implementation.

Inspector agents are incarnations of flaw detection modules, *Constructors* that of plan modification generating modules. The system additionally introduces so-called *Assistants* that provide shared inferences and services which are required by other agents. They propagate, for example, implications of temporal action information transparently into the ordering constraints [17]. *Strategies* finally implement planning strategy modules which control the search process by selecting plan modification steps for application to the plan.

Figure 1 shows the reference planning model for Panda which defines the agent interaction. One planning *cycle* corresponds to one iteration of a classical monolithic planning algorithm. It is divided into 4 phases, in which the agents are executed concurrently.

Phase 1: Assistants repeatedly infer information and post it on the blackboard until an inferential closure is reached (the "assistant cycle").

Phase 2: Inspectors analyze the current plan on the blackboard and report their detected flaws to the strategy and the constructors assigned to them by the trigger function.

Phase 3: Constructors compute all possible modifications for the received flaws and send them to the strategy.

Phase 4: The strategy compares all received results from inspectors and constructors and selects one plan modification to be executed on the current plan. One planning

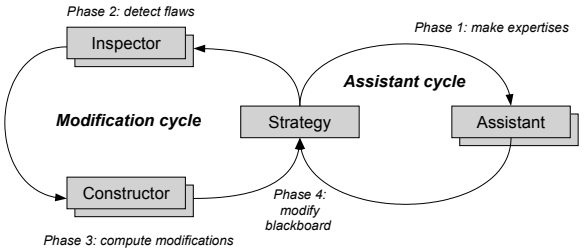

Figure 1: The reference planning process model for Panda.

cycle is hereby completed and the system continues with phase 1.

Phase transitions are performed by the strategy when all participating agents have finished execution and can thus be viewed as synchronization points within the process. The strategy modifies the plan until no more flaws are detected or until an inspector publishes a flaw for which no resolution is issued. In the first case the current plan constitutes a solution to the given planning problem, in the latter case the system has to backtrack and executes a different modification according to the strategy's stack of previous choices.

4 Knowledge-based Middleware

Core Components A direct implementation of the reference process model uses a single computational resource, processor, Java Virtual Machine, etc. For crisis management support, information must be however gathered from distributed and even mobile sources, the planning process requires a lot of computational power, solutions have to be distributed, etc. Thus scalability and distribution play key roles in the proposed system architecture, while maintaining the – simple but effective – reference process.

The main aspect in middleware systems like *application servers* is to hide the complexity of managing distributed objects and to provide abstract interfaces to the programmer. In other words, they make distribution issues *transparent*. Transparency in middleware systems covers location transparency, scalability transparency, access transparency, and concurrency transparency [7]. Scalability transparency, for example, hides the mechanisms how a middleware system scales in response to a growing load (allocating new computational resources on demand, etc.).

Panda builds on the application server JBoss [19], an implementation of the *Java 2 Enterprise Edition – J2EE* specification [20]. This standard provides a programming model for distributed applications, programming interfaces, and system policies. For this presentation we focus on three core technologies. The so-called *Enterprise Java Beans – EJBs* are the objects that are managed by an application server. They are the building blocks of a distributed J2EE application and all transparency aspects apply to them. A system wide directory service provides location and access transparency for all EJBs and services in the application server. Finally, a messaging service establishes asynchronous and location transparent communication between Java components (especially EJBs) beyond virtual machine boundaries. Panda uses such EJBs, for example, for modeling the blackboard and for connecting a client application to the Panda system (clients obtain an interface by querying JBoss's directory service).

The second core component for implementing all agent specific features is BlueJADE [5], an agent platform that is integrated into the JBoss service structure and that is compliant to standards defined by the Foundation for Intelligent Physical Agents (FIPA). This sub-system puts the agent lifecycle under full control of the application server, thus all distribution capabilities of the application server apply to the agent societies. It also provides standardized protocols on top of the J2EE message facility. Agents can therefore be spread transparently over several agent containers on different nodes in a network, including the migration of running agents between containers. It has to be noted, that this kind of distribution management differs from the corresponding middleware service: agent migration typically anticipates *pro-actively* the computation or communication load in a reasoning process, while middleware migration *reacts* on such load changes based on very low-level operating system specific sensors. It makes sense to provide both mechanisms in parallel, for example, to migrate in advance scheduling inspector agents which are known to require much computational resources onto dedicated compute servers. BlueJade's LEAP extension [3] adds support for ubiquitous computing. Agents are able to run even on mobile devices like PDAs, which provides Panda with convenient client interfaces for many application domains.

The knowledge representation and reasoning facilities which are used throughout the system constitute the third core component. During its development, the Panda framework required an increasing amount of knowledge that represents planning related concepts (flaw and modification classes, the trigger function), the system configuration (which agents to deploy), and the plan generation process itself (the reference process, including the backtracking procedure, etc.). Most of this knowledge is typically represented implicitly inside algorithms and data structures. To make it explicit and modifiable without touching the system's implementation, it must be extracted and represented in a common knowledge base which uses a representation formalism that is expressive enough to capture all modeling aspects on one side and that allows efficient reasoning on the other side. As a result of this, the system can be configured generically and that configuration can be verified on a higher semantic level. There is a large number of knowledge representation systems available on the market which promise to meet the requirements. The *DARPA Agent Markup Language – DAML* [13] has been chosen as the grounding representation formalism for this task, which combines the key features of description logics with Internet standards such as XML or RDF [14].

There exist powerful reasoners and libraries to integrate this language into applications, including mappings between the encoded knowledge and an object model. RACER [12] is Panda's description logic system to store knowledge and to reason about it. It has the essential capabilities that are required: a DAML codec, an efficient reasoning component, and a knowledge store based on a client-server architecture. The RACER-server is integrated via EJB proxies, hence all agents have transparent access to their personal request queue.

Auszug aus: Künstliche Intelligenz, Heft 1/2007, ISSN 0933-1875, BöttcherIT Verlag, Bremen, **www.kuenstliche-intelligenz.de/order**

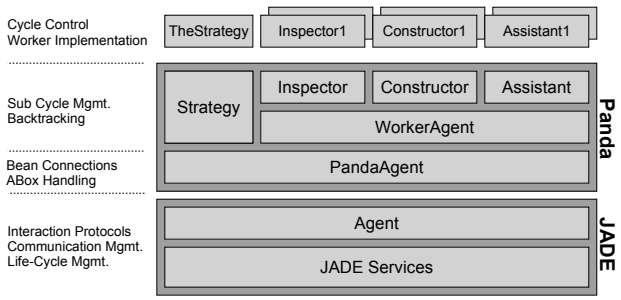

Figure 2: The logical layer structure of the agent framework.

By using DAML as the content language for the BlueJADE agent communication and also as the language for describing the system configuration and communication means, a homogeneous representation is realized in the Panda system.

The System Structure Two main classes of agents exist in the BlueJADE agent container. The first is the class of standard agents that come with the BlueJADE software package. They provide the FIPA infrastructure, BlueJADE specific communication services, etc. The role of *Gateway* agents, for example, is to mediate messages between BlueJADE agents and components outside the agent container. The Panda agents form the second class in the agent container. The `PandaAgent` class on its part is derived transitively from the BlueJADE agent base class `Agent` which provides the integration into the agent system (cf. Figure 2). It encapsulates low level data conversion and communication mechanisms. All agent types from the reference model are derived from `PandaAgent`.

The reference model in Fig. 1 omits the actual means for *calling* agents: (remote) calls are typically message based in distributed applications. From the container's point of view, agents in the BlueJADE agent container and the proxy EJBs communicate by using messages encoded in the agent communication language *FIPA-ACL* [10]. *ACL* is based on the speech-act theory, in which every message describes an action that is intended to be carried out with that message simultaneously, for example, requests like "compute detections". Such intentions are called *performatives* and *ACL* defines formal semantics for them [9]. They induce basic interaction protocols upon which more complex protocols like contract nets and auctions are built.

Besides parameters that are generally necessary for communication like sender, receiver, etc., *ACL* messages include parameters that describe the content that is intended for the receiving participant: the *language* the content is encoded in, the *domain* the content refers to, etc.

The Planning Process Figure 3 shows the refined model of the planning process that is implemented in the Panda system. The white colored states specify the life-cycle management of a planning session (initializing the process, starting planning, suspending it, etc.). Each state transition is labeled with the triggering *ACL* message and its originator: `sender:performative` followed by `action` or `proposition`. Senders can also be described by their class. `Worker` denotes the worker agent classes of inspectors, constructors, and assistants, for example. The same applies to propositions and actions: `Compute` denotes the action `Compute` and all subactions like `Inspect`, `Construct`, etc.

The planning process starts in a state in which all agents are deployed and send agreements for their initialization process. After that, the strategy informs all agents, that the system is initialized; this is where the reference model starts with phase 1: The assistants are requested to perform their inference on which they have to agree. After their processing (leaving the `assisting` state), the inspectors are requested to search for flaws (phase 2), and so on.

Most states in the figure are abstract in order to reduce complexity of the state automaton while maintaining a degree of granularity that allows the user to monitor the planning process. The state `backtracking`, for example, summarizes all possible sub-states that describe the interaction between a particular worker agent and the strategy while interrupting and setting back the agent.

The refined planning process model has two major improvements: First, it extends agent concurrency. The reference planning process model in Fig. 1 synchronizes the agent classes such that concurrency is only allowed within a particular phase. But the agents' execution can interleave in particular between phases 2 and 3, so that every constructor can start pro-actively its computations as soon as it received all flaws of all its α-assigned inspectors. The refined process model reflects this autonomy by a combined `inspecting&constructing` state in which some constructors are executed in parallel with detectors.

Second, optimized and more sophisticated reasoning techniques can be implemented with an enhanced backtracking procedure that allows assistants, inspectors, and constructors to maintain a local memory (for caching, etc.). Worker agents now participate actively in the backtracking process in order to keep backtracking consistent: they synchronize after an interrupt request by the strategy via agreement statements (cf. state transitions from `backtracking`). The restart of the system has to be negotiated afterwards to ensure that all agents have finished their local backtracking procedures.

With this mechanism, the extended concurrency pays off in terms of system performance, because the system is now able to backtrack immediately in early fail situations like "fast" inspectors without assigned constructors for publishing flaws. This is in fact the case for most of the inconsistencies that occur in the constraint sets: they are unrepairable.

Ontology-based Components DAML is used as the universal representation formalism to describe and share knowledge in the Panda system, ranging from flaw communication to system state transitioning. It is one of the emerging standards in the Semantic Web community for representing and communicating knowledge [13]. It ensures interoperability with third-party systems like RACER and forms the basis for communicating knowledge among agents. Most important, it enables knowledge to be represented in a uniform, explicit, and declarative manner, so the system becomes more robust, flexible, and maintainable.

In order to be able to use DAML as a content language for *ACL* messages with their speech act structure, at least ac-

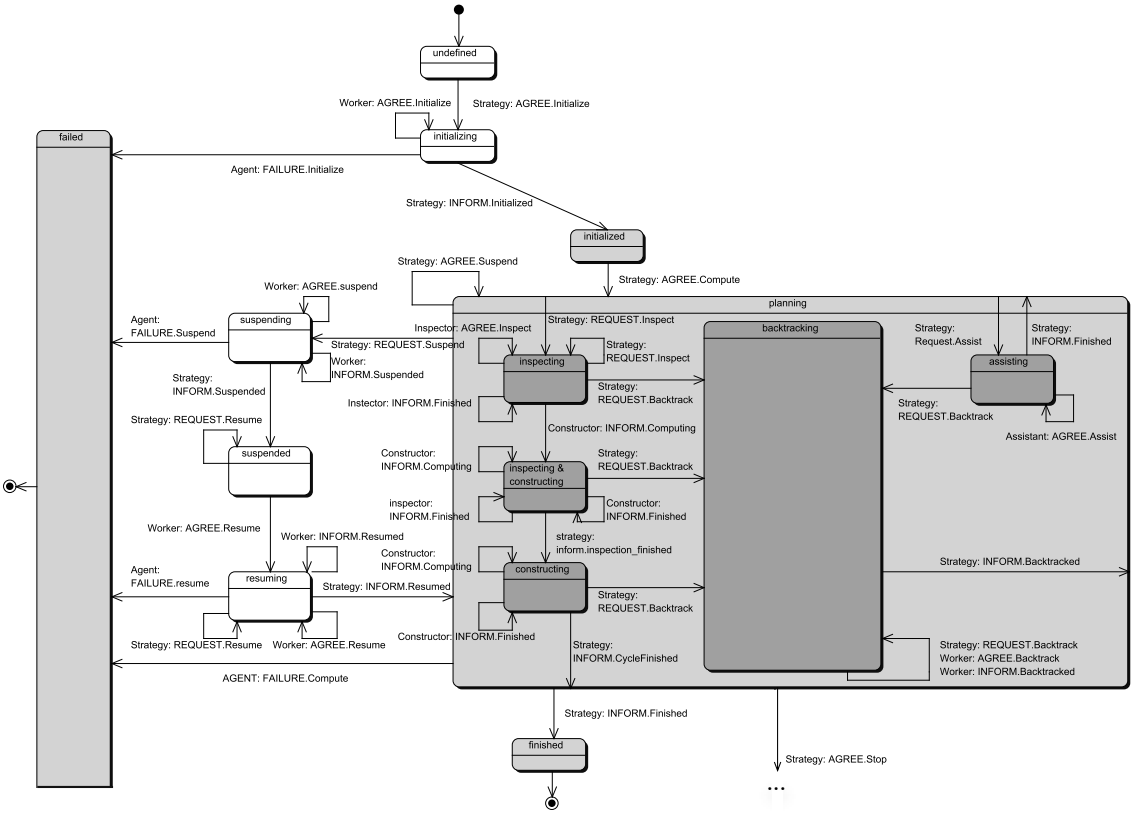

Figure 3: The refined Panda planning process model.

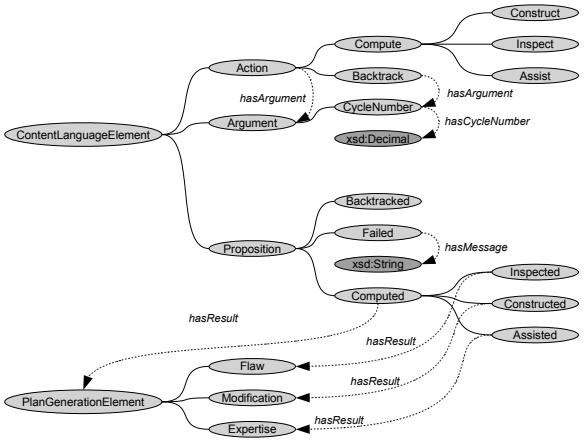

Figure 4: The content language ontology.

tions and propositions must be representable [15]. This is sufficient for the needs of Panda. Figure 4 shows the ontology that provides the concepts to enable DAML-based communication (property cardinalities have been omitted for clarity).

Actions can have arguments, for example, the action sub-concept `Backtrack` must come with a `CycleNumber` whose value is represented as the XML-schema type `decimal`. In Panda, every argument of an action is modeled in the ontology in order to give it formal semantics. Therefore, in contrast to [15], the arguments' order does not have to be considered. Propositions are currently only used to represent action results. The result sub-concept `Computed` has an argument of type `PlanGenerationElement`: For example, a `Constructed` proposition has a `Modification` element as a result. The content of an *ACL* message is represented by instances of the Panda system ontology embedded in a DAML document. Encoding and decoding of DAML message content is performed by third-party libraries which synchronize the agents' object models with the ontology.

DAML plays its second key role in the automated configuration of the agent container (Figure 5 shows the underlying ontology). The configuration process is composed of two steps. First, the agents that are part of the planning process are instantiated. The client proxy EJB initializes the system start-up and uses the RACER reasoner to derive the leaf concepts of `PandaAgent` and to determine the implementation assignments `ImplementationElements` of the worker agents. In Figure 5, for example, the assigned implementation for agent `Inspector1` is an instance of the Java class `panda.jade.agent.Inspector1Impl`. After their creation, the Panda agents insert their descriptions as instances into the system ontology so that the ontology reasoner can keep track of deployed agent instances.

The second configuration step is to implement the trigger function α by establishing communication channels between the respective agents instances. RACER is used by each Panda agent on startup to derive its communication links from and to other agents. This is done by using the system configuration ontology to derive the dependencies between agents from defined dependencies between the flaws and modifications: The ontology specifies, which agent instance implements which type of `Inspector` and `Constructor` agent. The RACER reasoner derives from this information, which flaw

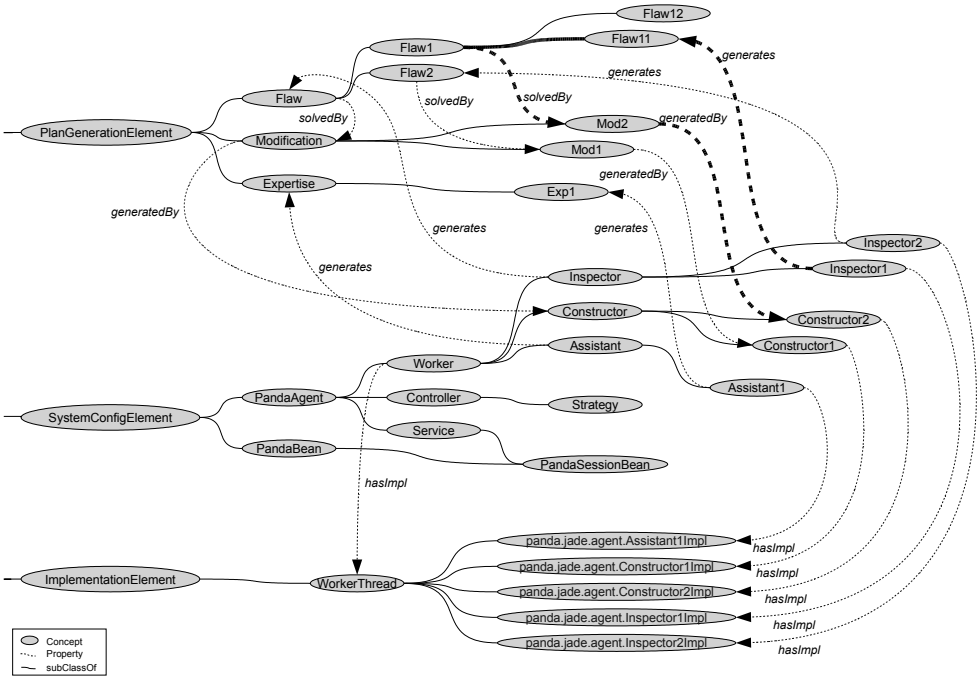

Figure 5: The system configuration ontology.

and modification types the agent instances will generate. If the system configuration ontology includes an α-relationship between them (`solved-by`), the agent instances' communication ports are linked.

Based upon the subsumption capabilities that come with DAML and description logics, it is even possible to exploit sub-class relationships between `PlanGenerationElements` (illustrated by the bold printed concept connections in Fig. 5). An example for a modification class hierarchy are ordering relation manipulations with sub-classes *promotion* and *demotion*. Regarding flaws, the system ontology distinguishes *primitive* open preconditions and those involving reasoning over *decomposition axioms* [2].

A knowledge based configuration offers even more benefits: Imagine a less informed configuration mechanism, say, reading a respective file in XML format, that holds the descriptions on the agents to be loaded and the message links to be established between them as a representation of the trigger function α. Configuration verification can then only be based on type checking by the Virtual Machine's class loader. In the presented architecture, the system model can be checked on startup for specific properties of the configuration in a verification step of the planning process in state `initializing` (Fig. 3). Examples are inconsistencies like constructors without assigned flaws (and consequently no inspector) and potential mis-configurations like flaw and modification classes for which no generating agent implementations are provided, etc.

5 Related Work

There are two major agent-based planning architectures on the market. In the O-Plan system [22], a blackboard is examined by combined inspector and constructor modules that write their individually highest ranked flaw on the agenda of a search controller. This controller selects one agenda entry and triggers the respective module to perform its highest prioritized modification. O-Plan has been extended by a workflow-oriented infrastructure with a plug-in mechanism that serves as an interface to various (application tailored) tools [21]. The planning engine itself is a monolithic system structure. The Multi-agent Planning Architecture [23] is based on a generic agent-based distributed problem solving approach. It executes an agent society in which designated coordinators decompose the planning problem into sub-problems which are solved by subordinated groups of agents that may again decompose the problem again. Individual solutions are returned to the associated managers which are responsible for synthesizing global solutions. Communication of queries and results is based on the KQML formalism which is quite similar to *FIPA-ACL*.

The SIADEX architecture [6] uses XML-RPCs for building a distributed planning environment, that is accessible via standardized HTTP and Java protocols. The architecture decouples a monolithic planning server, knowledge base management, and execution monitoring.

A representative for an application framework for building planning applications is Aspen [11]. It provides planning-specific data infrastructure, supportive inference mechanisms, and algorithmic templates, in order to facilitate rapid development of non-distributed planning applications "out-of-the-box".

6 Conclusions

We have presented a novel architecture for planning systems. It relies on a formal account of hybrid planning, which allows to decouple flaw detection, modification computation, and search control [18]. Planning capabilities like partial order planning and abstraction techniques can easily be com-

bined by orchestrating respective elementary modules and an appropriate strategy module – in particular flexible strategy modules. The implemented system can be employed as a platform to implement and evaluate various planning methods and strategies. It can be easily extended to additional functionality, like integrated scheduling [16, 17] and probabilistic reasoning [1], without jeopardizing system consistency through interfering activity.

This work has investigated three main aspects regarding the Panda planning system architecture. Using knowledge representation and inference techniques extends the capabilities of the system significantly. The verification of system configurations can be performed on an abstract level and the system becomes more flexible and safely manageable the more hard-coded knowledge is extracted and described declaratively. With the use of application server technology and standardized communication protocols, Panda has laid the foundation for a distributed system that is able to address real-world application scenarios in an adequate manner.

We plan to deploy this system as a central component in projects for assistance in telemedicine applications as well as for personal assistance in ubiquitous computing environments.

References

[1] S. Biundo, R. Holzer, and B. Schattenberg. Project planning under temporal uncertainty. In *Planning, Scheduling, and Constraint Satisfaction: From Theory to Practice*, pages 189–198. IOS Press, 2005.

[2] S. Biundo and B. Schattenberg. From abstract crisis to concrete relief – A preliminary report on combining state abstraction and HTN planning. In *Proceedings of the 6th European Conference on Planning (ECP-01)*, 2001.

[3] G. Caire. LEAP users guide, 2005. http://jade.tilab.com/doc/LEAPUserGuide.pdf.

[4] L. Castillo, J. Fdez-Olivares, and A. González. On the adequacy of hierarchical planning characteristics for real-world problem solving. In *Proceedings of the 6th European Conference on Planning (ECP-01)*, 2001.

[5] D. Cowan and M. Griss. Making software agent technology available to enterprise applications. Technical Report HPL-2002-211, Software Technology Laboratory, HP Laboratories, 2002.

[6] M. de la Asunción, L. Castillo, J. Fdez.-Olivares, O. García-Pérez, A. González, and F. Palao. Knowledge and plan execution management in planning fire fighting operations. In *Planning, Scheduling, and Constraint Satisfaction: From Theory to Practice*, pages 159–168. IOS Press, 2005.

[7] W. Emmerich. *Engineering Distributed Objects*. Wiley, 2000.

[8] T. A. Estlin, S. A. Chien, and X. Wang. An argument for a hybrid HTN/operator-based approach to planning. In *Proceedings of the 4th European Conference on Planning (ECP-97)*, volume 1348 of *LNAI*, pages 182–194. Springer, 1997.

[9] FIPA - Foundation for Intelligent Physical Agents. *FIPA-ACL Communicative Act Library Specification*, 2002. http://www.fipa.org/specs/fipa00037/SC00037J.pdf.

[10] FIPA - Foundation for Intelligent Physical Agents. *FIPA-ACL Message Structure Specification*, 2002. http://www.fipa.org/specs/fipa00061/SC00061G.pdf.

[11] A. Fukunaga, G. Rabideau, S. Chien, and D. Yan. Towards an application framework for automated planning and scheduling. In *Proceedings of the 1997 International Symp. on AI, Robotics & Automation for Space*, 1997.

[12] V. Haarslev and R. Möller. Description of the racer system and its applications. In *Working Notes of the 2001 International Description Logics Workshop (DL-2001)*, volume 49 of *CEUR Workshop Proceedings*, 2001.

[13] I. Horrocks, F. Harmelen, and P. Patel-Schneider. DAML+OIL Specification (March 2001), 2001. http://www.daml.org/2001/03/daml+oil-index.html.

[14] F. Manola and E. Miller. RDF primer, 2004. http://www.w3.org/TR/rdf-primer/.

[15] M. Schalk, T. Liebig, T. Illmann, and F. Kargl. Combining FIPA ACL with DAML+OIL - a case study. In *Proceedings of the Second International Workshop on Ontologies in Agent Systems*, 2002.

[16] B. Schattenberg and S. Biundo. On the identification and use of hierarchical resources in planning and scheduling. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS-02)*, pages 263–272. AAAI, 2002.

[17] B. Schattenberg and S. Biundo. A unifying framework for hybrid planning and scheduling. In *Proceedings of the 29th German Conference on Artificial Intelligence (KI 2006)*, LNAI. Springer, 2006. to appear.

[18] B. Schattenberg, A. Weigl, and S. Biundo. Hybrid planning using flexible strategies. In *Proceedings of the 28th German Conference on Artificial Intelligence (KI 2005)*, volume 3698 of *LNAI*, pages 258–272. Springer, 2005.

[19] S. Stark. *JBoss Administration and Development*. JBoss Group, LLC, second edition, 2003. JBoss Version 3.0.5.

[20] Sun Microsystems. *Simplified Guide to the Java 2 Platform, Enterprise Edition*, 1999. http://java.sun.com/j2ee/white/j2ee$_{guide.pdf}$.

[21] A. Tate. Intelligible AI planning. In *Proceedings of the 20th British Computer Society Special Group on Expert Systems International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, pages 3–16. Springer, 2000.

[22] A. Tate, B. Drabble, and R. Kirby. O-Plan2: An architecture for command, planning and control. In *Intelligent Scheduling*, pages 213–240. Morgan Kaufmann, 1994.

[23] D. Wilkins and K. Myers. A multiagent planning architecture. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 154–163. AAAI, 1998.

Contact
Prof. Dr. Susanne Biundo
Department of Artificial Intelligence
Ulm University, 89069 Ulm, Germany
Tel.: +49 (0)731-5024121
http://www.informatik.uni-ulm.de/ki/

Susanne Biundo is a Professor of Computer Science at the University of Ulm. Her research interests include Artificial Intelligence Planning and Scheduling, Automated Reasoning, and Multiagent Technology. She was the director of PLANET, the European Network of Excellence in AI Planning, from 1998–2003.

Bernd Schattenberg is a research assistant at the department of Artificial Intelligence of the University of Ulm. His main research interests are the integration of planning and scheduling methods and their application to complex real-world scenarios.

Steffen Balzer studied computer science at the University of Ulm, where he worked as a research assistant in the field of Semantic Web Services. He is now engaged as a consultant for Service Oriented Architectures at IT-Informatik GmbH.