

The Struggle for Reuse and Domain Independence: Research with TÆMS, DTC, and JAF.

Thomas Wagner
Computer Science Department
University of Maine
wagner@umcs.maine.edu

Bryan Horling
Computer Science Department
University of Massachusetts
bhorling@cs.umass.edu

ABSTRACT

TÆMS, Design-to-Criteria agent scheduling, and the Java Agent Framework are important aspects of our research in generalized agent components. These technologies have been used in over a dozen different research efforts, resulting in some insight into the pros and cons of generalized control and providing some anecdotal evidence that may be useful to other researchers.

1. INTRODUCTION

One major facet of our work is the attempt to create generalized domain independent control components for autonomous intelligent agents. We have been pursuing this line of research for nearly ten years and the components that we have constructed have been used on approximately a dozen different research projects. The motivation behind domain independence is that it enables us to reuse the control components on different applications with few modifications, though this has been successful to varying degrees. In the following sections we describe the components and discuss their strengths and weaknesses, and issues that have arisen during the course of our research.

It is important to emphasize that this research pertains to complex problem solving agents, e.g. the BIG Information Gathering Agent [20] and [15, 4, 14], where the agents are situated in an environment, able to sense and effect, and have explicit representations of candidate tasks and explicit representations of different ways to go about performing the tasks. Additionally, tasks are quantified or have different performance characteristics and, following in the thread of complex problem solving [10, 8, 22] there are relationships between the tasks. This means that there is a high degree of interconnectivity in agent problem solving and that reasoning about what the agent should be doing, with whom to coordinate, etc., is always a non-trivial process.

In our work we achieve domain independence by abstracting away from the details of a particular domain and reasoning about it via a model of the agent's problem solving process. The modeling framework we use is called TÆMS [6, 18]. TÆMS task structures resemble complex and/or graphs or HTNs, i.e., TÆMS is a hierarchical decomposition framework. Notable features of TÆMS models include the explicit representation of alternative dif-

ferent ways to perform tasks, the explicit representation and quantification of task interactions, and the characterization of primitive actions in terms of their quality, cost, duration, and uncertainty. A sample TÆMS task structure is shown in Figure 2 (the figure is discussed in greater detail later). Once an agent's options are modeled in TÆMS, our domain independent control problem solving components can decide which actions an agent should take, what resources to use, which tasks to coordinate with other agents, and how the agent can meet real-time deadlines and resource constraints. The core of this ability is the Design-to-Criteria [29, 27, 26, 23] agent scheduler – it is the TÆMS analysis expert. When multi-agent systems are constructed, a coordination module is added to the agent and often the module is GPGP [7, 17] or one of its descendants [25]. Though we have used proprietary components for this as well [16, 11].

A natural question is where do the TÆMS models come from? In our work TÆMS task structures are often hand crafted though the general idea is that a domain expert, e.g., a blackboard problem solver or a planner, should translate its problem solving options into TÆMS. This model was used successfully in the BIG information gathering agent [20]. In some recent projects, e.g., the TripBot [30], the domain expert plans directly in TÆMS. The architecture of the TripBot is shown in Figure 1. In the TripBot, a query enters the system, is expanded / enhanced using WordNet [3] and passed to a TÆMS information gathering planner (called the "capability assessor" for a variety of reasons), which is a new TÆMS artifact. The agent's options are then passed to the Design-to-Criteria (DTC) scheduler along with a specification of the agent's current objective function, e.g., deadlines, solution characteristics, etc. DTC then evaluates the agent's options, in light of its objectives, and determines a course of action for the agent. The resulting schedule is then executed and the agent reschedules and replans on failure as necessary. This is how TÆMS and DTC are generally used in an agent.

When a coordination or communication module is added DTC serves as the oracle for the coordination module – enabling the module to understand the implications of forming commitments with other agents. For example, if agent α should commit to agent β to perform task T_1 by time 10, α needs to be able to evaluate the cost of offering that commitment to β in terms of α 's local problem solving.

This domain-independence coupled with a component ("plug-and-play") approach to agent construction has proved its merit by enabling us to reuse the technology in many different application domains, e.g., the BIG information gathering agent [21, 20], the Intelligent Home project (IHome) [16], the DARPA ANTS real-time agent sensor network for vehicle tracking [11, 14], distributed hospital patient scheduling [6], distributed collaborative design [9],

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2000 ACM 0-89791-88-6/97/05 ..\$5.00

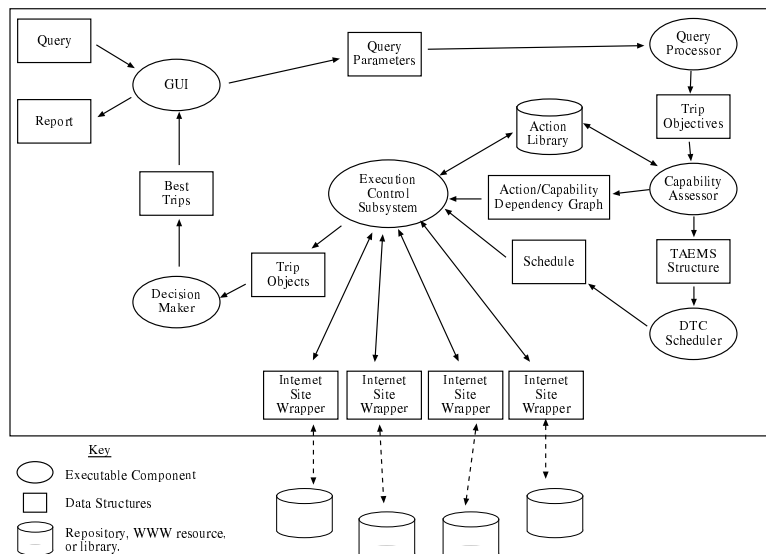


Figure 1: One Instantiation: The TripBot Agent Architecture

process control [31], the TripBot [30], agent diagnosis [1, 13], and others. However, in many of these projects modifications to the artifacts were required and we became aware of certain design decisions that affected our ability to move to a new application, change the control flow within the agent, or to expand the TÆMS task modeling language.

A component hereto unmentioned is the Java Agent Framework [12], or JAF, which is a fairly recent comer to the scene but which has provided the glue by which the components are integrated on projects like the IHome [16] and the ANTS real-time agent sensor network for vehicle tracking [11, 14]. Like Decker’s DECAF [5], the goal of JAF is to provide the framework that integrates the different control components and supports their interaction. Another related component used in both IHome and ANTS is the multi-agent system simulator [24] that enables different JAF agents to “execute” TÆMS primitive actions in a controlled environment.¹

In this paper we focus on TÆMS, the Design-to-Criteria scheduler, and the Java Agent Framework and the experiences we have gained from working with these technologies. The rationale for this emphasis is that TÆMS and DTC are among the components that have been in service the longest and used in the widest range of application settings. By the same token, the lessons learned from the java agent framework are widely applicable to the agents community because at some level, we are all faced with the problem of how to integrate our components to support flexibility and functionality.

It is also worth noting that these technologies have been available online for distribution in the past and we are currently planning on releasing them into the public again in the near term.

2. TÆMS TASK MODELS

2.1 What TÆMS Is

TÆMS (Task Analysis, Environment Modeling, and Simulation) is a domain independent task modeling framework used to describe and reason about complex problem solving processes. TÆMS models are hierarchical abstractions of problem solving processes that

¹Obviously for projects like the TripBot that operate in a real domain the simulation environment is not required, though it could be used for debugging TÆMS related control activities.

describe alternative ways of accomplishing a desired goal; they represent major tasks and major decision points, interactions between tasks, and resource constraints but they do not describe the intimate details of each primitive action. All primitive actions in TÆMS, called *methods*, are statistically characterized via discrete probability distributions in three dimensions: quality, cost and duration. Quality is a deliberately abstract domain-independent concept that describes the contribution of a particular action to overall problem solving. Duration describes the amount of time that the action modeled by the method will take to execute and cost describes the financial or opportunity cost inherent in performing the action. Uncertainty in each of these dimensions is implicit in the performance characterization – thus agents can reason about the certainty of particular actions as well as their quality, cost, and duration trade-offs. The uncertainty representation is also applied to task interactions like enablement, facilitation and hindering effects,² e.g., “10% of the time facilitation will increase the quality by 5% and 90% of the time it will increase the quality by 8%.” The quantification of methods and interactions in TÆMS is not regarded as a perfect science. Task structure programmers or problem solver generators *estimate* the performance characteristics of primitive actions. These estimates can be refined over time through learning and reasoners typically replan and reschedule when unexpected events occur.

To illustrate, consider Figure 2, which is a conceptual, simplified sub-graph of a task structure emitted by the BIG [19] information gathering agent; it describes a portion of the information gathering process. The top-level task is to construct product models of retail PC systems. It has two subtasks, *Get-Basic* and *Gather-Reviews*, both of which are decomposed into methods, that are described in terms of their expected quality, cost, and duration. The *enables* arc between *Get-Basic* and *Gather-Reviews* is a non-local-effect (nle) or task interaction; it models the fact that the review gathering methods need the names of products in order to gather reviews for them. Other task interactions modeled in TÆMS include: *facilitation*, *hinder-*

²Facilitation and hindering task interactions model soft relationships in which a result produced by some task may be beneficial or harmful to another task. In the case of facilitation, the existence of the result, and the activation of the nle generally increases the quality of the recipient task or reduces its cost or duration.

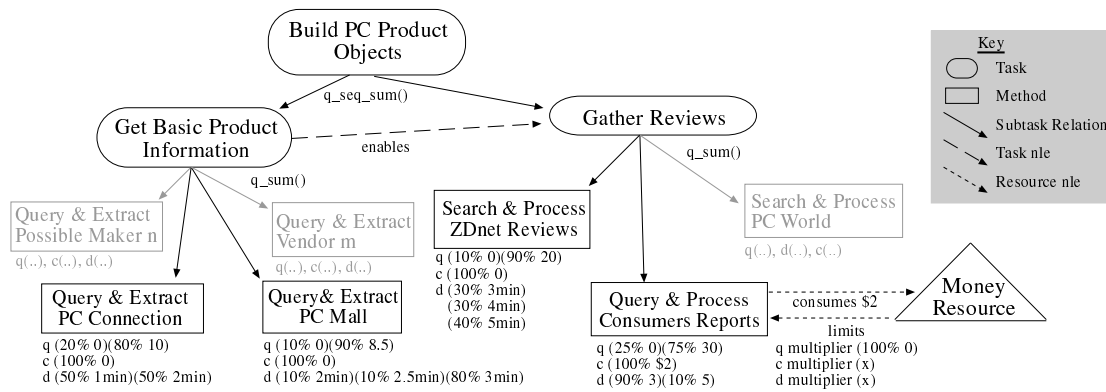


Figure 2: Simplified Subset of an Information Gathering Task Structure

ing, bounded facilitation, sigmoid, and disablement. Task interactions are of particular interest to coordination research because they identify instances in which tasks assigned to different agents are interdependent – they model, in effect, implicit joint goals or joint problem solving activity. Coordination is motivated by the existence of these interactions.

Returning to the example, *Get-Basic* has two methods, joined under the $sum()$ quality-accumulation-function (qaf), which defines how performing the subtasks relate to performing the parent task. In this case, either method or both may be employed to achieve *Get-Basic*. The same is true for *Gather-Reviews*. The qaf for *Build-PC-Product-Objects* is a $seq_sum()$ which indicates that the two subtasks must be performed, in order, and that their resultant qualities are summed to determine the quality of the parent task; thus there are nine alternative ways to achieve the top-level goal in this particular sub-structure. In general, a TÆMS task structure represents a family of plans, rather than a single plan, where the different paths through the network exhibit different statistical characteristics or trade-offs.

2.2 Pros, Cons, and What We Would Do Differently

One of the major strengths of TÆMS is that it is generally very expressive. TÆMS was designed to model the problem solving activities of complex cooperative blackboard problem solvers and it is very good at modeling tasks and interrelationships. However, as always, there is a trade-off between representational strength and tractability. Certain features of TÆMS, e.g., soft task interactions that enable the performance of one task to affect the duration of another, make reasoning/scheduling TÆMS task structures very difficult. In our opinion, one question to ask when developing a modeling or representational structure is how tractable the resulting model will be. If a model is expressive but intractable, research is limited either to toy sized problem spaces or some complex (and usually approximate) artifact like the Design-to-Criteria scheduler is required. If the research interest does not pertain to scheduling or complex analysis, then this should be avoided if possible.

Documentation with TÆMS has proven to be another issue that should have been addressed sooner. Prior to the TÆMS whitepaper [18], graduate students were handed a dissertation or a small stack of research papers and forced to generate the current intellectual model of TÆMS in a bottom-up fashion. A natural question is why? The answer is that research is sometimes a rapidly moving target and that, as all basic researchers know, there are times that resources are scarce. The lack of such documentation did not become

an issue until the number of new researchers working in TÆMS (or attempting to do so) grew above a certain threshold. There is clearly a time at which it is important to “sure up” one’s footing and get everyone on the same page – possibly to the detriment of the pace at which the basic research progresses. This documentation, and related tutorial-esque discussions between members of the group, have clearly paid dividends as the number of researchers now able to discuss problem solving via TÆMS has grown significantly – spreading to at least four universities and encompassing at least five faculty members and twenty graduate students. While this is attributable to other concurrent happenings, e.g., development and sharing of infrastructure like DTC and JAF, the existence of tutorial style documentation helped significantly.

The development of a usable and sharable TÆMS modeling implementation has also paid significant dividends. On or about 1994 there was a lisp version of TÆMS that was integrated with a simulation environment and only its original creators were able to get it to turn over. A new implementation in C++ for use with the DTC scheduler was the first implementation in which TÆMS was a stand alone artifact. However, by the time the next TÆMS artifacts were developed, around 1996, java had become the language of choice for this class of development and a new implementation was needed. While the C++ and lisp versions of TÆMS are still in use (the C++ version is tied to DTC and thus used in nearly all projects) the bulk of the researchers at UMASS rely on a common implementation of TÆMS in java. Support tools for TÆMS have also paid real dividends and these are, in a sense, combined with the java version of TÆMS. Simple ways to create, manipulate, view, and store task structures are required for widespread use. Again, this concept is obvious to anyone making an artifact intended for general use, but, in basic research there is generally some give and take between implementation accouterments and progress on the research front.

Is there anything wrong with TÆMS? Yes. In part because the way it is being used has evolved and in part because for some research it is necessary to get closer to the details, TÆMS has some oddities and inconsistencies. One notable example is the difference between the resource models supported in the java implementation of TÆMS and those supported by the C++/DTC implementation. DTC adheres to the basic property of TÆMS that methods are black boxes and there is no correlation between, for example, quality and duration or resource consumption and duration. There are situations, however, where it is more intuitive to correlate time and resource consumption (though there are cases where it is not, also). Because of this and the particular applications for which

the java side was being used, we now have multiple different resource models that are only loosely compatible. Another example of inconsistencies can be found in the TÆMS quality accumulation functions – some qafs impose ordering on the subtasks, some do not, and some specify whether or not particular subtasks must be performed, and some do not. The original conceptualization of TÆMS did not specify orderings but for several domains we found the modeling structure insufficient and it was extended. What is the moral of the story? There is value in application but even the best conceived artifact is likely to be pulled and stretched when it is actually used.

Other problems or reservations with TÆMS exist. For example, by being abstract, it needs to be coupled with detailed information to be used in agents operating in real environments, e.g., the dependency specification used in the TripBot [30]. Another issue is the way quality propagates through the network and the limitation, some of which is implementational, to reasoning only in terms of quality, cost, duration, and uncertainty in each of these dimensions (in contrast to our new *MQ* framework [28]). However, most of these issues fall into category of basic research questions.

3. DESIGN-TO-CRITERIA SCHEDULING: LOCAL AGENT CONTROL

3.1 What DTC Is

The Design-to-Criteria (DTC) scheduler is the agent’s local expert on making control decisions. The scheduler’s role is to consider the possible domain actions enumerated by the domain problem solver and choose a course of action that best addresses: 1) the local agent’s objectives or goal criteria (its preferences for certain types of solutions), 2) the local agent’s resource constraints and environmental circumstances, and 3) the non-local considerations expressed by the agent’s (optional) coordination module. The general idea is to evaluate the options in light of constraints and preferences from many different sources and to find a way to achieve the selected tasks that best addresses all of these.

The scheduler understands the agent’s situation and objectives via TÆMS task structures and reasons about different possible courses of actions via TÆMS. Scheduling problem solving activities modeled in the TÆMS language has four major requirements: 1) to find a set of actions to achieve the high-level task, 2) to sequence the actions, 3) to find and sequence the actions in soft real-time, 4) to produce a schedule that meets dynamic objective criteria of the agent. The reason we require soft real-time is that DTC is designed for open environments where unpredicted change is commonplace. When change occurs, the agent reinvokes DTC to decide on an appropriate course of action.

TÆMS models multiple approaches for achieving tasks along with the quality, cost, and duration characteristics of the primitive actions, specifically to give agent’s flexibility in problem solving. This is how TÆMS agents can respond to new situations and how they can custom tailor their problem solving for different situations. A classic example being to trade-off solution quality for shorter duration when time is limited. DTC is the agent’s trade-off and control expert. In contrast to classic scheduling problems, the TÆMS scheduling objective is not to sequence a set of unordered actions but to *find* and *sequence* a set of actions that *best* suits a the agent’s current objectives.

Design-to-Criteria scheduling requires a sophisticated heuristic approach because of the scheduling task’s inherent computational complexity ($\omega(2^n)$ and $o(n^n)$) it is not possible to use exhaustive search techniques for finding optimal schedules. Furthermore, the

deadline and resource constraints on tasks, plus the existence of complex task interrelationships, prevent the use of a single heuristic for producing optimal or even “good” schedules. Design-to-Criteria copes with these explosive combinatorics through approximation, criteria-directed focusing (goal-directed problem solving), heuristic decision making, and heuristic error correction. The algorithm and techniques are documented more fully in [29, 27, 26, 23].

3.2 The Good, the Bad, and the Ugly

DTC is an extremely complex artifact encompassing around 50,000 lines of C++ code. It is fast for what it does – scheduling large task structures (having fifty primitive actions) in less than 10 seconds and performing hundreds of thousands of probability distribution combination operations in that time (on a basic PIII-600 class machine running linux). However, there are applications for which DTC running in exhaustive scheduling mode will run in half the time as DTC running in its normal mode which is designed to cope with high order combinatorics from many different sources. In some cases, for some task structures, doing it the brute strength way is actually faster and more effective. Relatedly, for applications that have particular regular properties, e.g., using a single TÆMS task structure with different bindings on the leaves, custom scheduling solutions can be developed that will outperform DTC both in terms of time and solution quality. The thought here is that domain independence in control is a hard problem and there are always performance trade-offs involved. Generality, by definition, means that artifacts have to address the hardest class of problems possible for a given problem instance.

Relatedly, DTC was written to be fast and some of the optimizations have proven obstacles when TÆMS was modified or changed in very particular ways. For example, the addition of the *sigmoid()* quality accumulation function meant that the scheduler had to track new and different information. Similarly, the addition of TÆMS qafs that impose orderings required some implementational acrobatics. If the scheduler had been designed less for speed from the beginning it would have been more easily extended. It is unclear if the scheduler should have been designed differently, as it has thus far been extendable to meet most of our needs, but, in our opinion, code optimization should only be applied to mature research technologies unless the artifact poses significant bottlenecks.

One of the major wins in DTC has been its encapsulation. DTC is stateless and obtains all of its information via input files and produces everything the client needs via output files. Written in C++, this stateless approach means that it is literally a stand-alone executable that clients invoke when needed. The one caveat with this model is that it requires the process-starting-overhead of the OS and cannot be invoked via native function calls from Java or lisp. Prior to DTC our scheduling technology was tightly coupled with the execution subsystem and assumed that it would have the ability to monitor task performance directly. In general, the separation of DTC and the movement of the agent’s problem solving state to DTC’s input has paid great dividends. Evidence of this includes the variety of ways that it has been used in our research, including other scheduler enhancements that build on top of DTC as an external clients, e.g, work in schedule caching and partial-ordering [11].

Related to DTC’s encapsulation was the construction of a human readable textual I/O format for DTC. Referred to as *t-TÆMS* by project members, the input to DTC is a textual representation of a TÆMS task structure plus a textual representation of the agent’s objective function and constraints like deadlines, resource limitations, etc. Given the input, DTC schedules and produces multiple

different output files. One of which is a detailed t-TÆMS schedule file that contains a ranked set of detailed schedules for the agent that includes DTC's expectations about task performance and the state of problem solving as tasks are executed. This information can then be used to determine when it is necessary to reinvoke DTC and re-plan/reschedule. DTC also produces a more human friendly schedule representation and a simple schedule description file for clients that do not wish to implement t-TÆMS scheduler parsers. The lesson learned here is that, obviously, good interfaces are important, but, also that textual representations often provide important for versatility and for human debugging.

Simplicity to the client has been another real design pay-off in DTC. While DTC is highly configurable in terms of the types of pruning and focusing it does, the types of analysis it does, the way it approaches resources, scheduling, analysis, and the way it evaluates probability distributions and uncertainty, most clients never need to customize these features. Thus, while most of them are accessible either via command line arguments or via the t-TÆMS input file, DTC does not require the client to specify anything himself. Instead DTC is configured with a set of default options that, in general, yield good performance. It is worth noting that in an ideal world DTC would classify problem instances and set defaults on a learned basis, but, right now it does not have this functionality. The important concept here is that no matter how complex one's artifact may be, most users or clients have little or no interest in having to understand a large set of research questions to use the artifact. If reuse is a goal, good defaults and a very simple interface is a clear win.

One of the caveats of reuse has also come to light with DTC. Reuse requires support. Period. As artifacts are applied to new projects, they are also used in different ways and in different environments. This can highlight simple bugs but also lead to larger support issues like having to modify the technology or to explain in detail why the artifact performs in certain ways. Quite often with research technologies these explanations are non-trivial because of the issues to which they relate. Support outside of a research lab is an obvious result of distribution, however, the support burden within a lab of a member whose technology is widely used should be recognized and explicitly addressed.

4. JAVA AGENT FRAMEWORK

The underlying technology of our Java Agent Framework (JAF) uses component-based technology designed to promote reuse and extension. Developers can use its plug and play interface to quickly build agents using existing generic components, or to develop new ones. The JAF architecture consists of two parts, a set of design conventions and a number of generic components. The design conventions provide guidance to the developer, which attempt to facilitate the creation, integration and reuse of newly written components. The generic components form a stable base for the agent, which the developer can use or extend as needed. For instance, a developer may require planning, scheduling and communication services in their agent. In this case, generic scheduling and communication components exist, but a domain-dependent planning component is needed. Additionally, the characteristics of the generic scheduling component do not satisfy all the developer's needs. The JAF design conventions provide the developer with guidance to create a new planning component capable of interacting with existing components without unduly limiting its design. A new scheduling component can be derived from the generic one to implement the specialized needs of their technology, while the communication component can be used directly. All three can easily interact with one another as well as other components in the agent, maximizing

code reuse and speeding up the development process.

JAF builds upon Sun's Java Beans model by supplying a number of facilities designed to make component development and agent construction simpler and more consistent. A schematic diagram for a typical JAF component can be seen in figure 3. As in Java Beans, events and state data play an important role in some types of interactions among components. Additional mechanisms are provided in JAF to specify and resolve both data and inter-component dependencies. These methods allow a component, for instance, to specify that it can make use of a certain kind of data if it is available, or that it is dependent on the presence of one or more other components in the agent to work correctly. A communications component, for example, might specify that it requires a local network port number to bind to, and that it requires a logging component to function correctly. These mechanisms were added to organize the assumptions made by component developers - without such specifications it would be difficult for the designer to know which services a given component needs to be available to function correctly. More structure has also been added to the execution of components by breaking runtime into distinct intervals (e.g. initialization, execution, etc.), implemented as a common API among components, with associated behavioral conventions during these intervals. Individual components will of course have their own, specialized API, and "class" APIs will exist for families of components. For instance a family of communication components might exist, each providing different types of service, while conforming to a single class API that allows them to easily replace one another.

JAF has been used successfully in several domains. It was originally conceived and developed to evaluate multi-agent system survivability within the MASS simulator [24]. Later, additional agents were developed in JAF within the IHome project [16], which looked at how multi-agent systems could play a role in an intelligent home environment. JAF agents were also augmented with a diagnosis component in IHome [13], and a Producer-Consumer-Transporter domain [2], to study the role diagnosis can play in dynamically adapting organizational design in response to environmental change. Most recently, JAF has been deployed in a distributed sensor network environment [14] where agents must organize to gather the sensor data required to track moving targets. In this last project, these same agents were also successfully migrated from a simulated environment to an actual physical system using Doppler radar sensors and moving targets. Over the course of these projects, roughly 30 different JAF components have been developed.

We have found the JAF framework relatively easy to use, once an initial learning curve is passed. In each of the projects mentioned above, roughly two-thirds to three-quarters of each agent were comprised of existing, reused code. In the cases where existing components could not operate in the given environment, as when agents are moved to a different simulator or into a physical system, only relatively minor extensions were required to low-level components (e.g. communication or execution), while the higher level components worked unchanged. The event-based interaction system permits components to be easily added and removed, while retaining the ability to react to actions performed by other components. State-based interaction takes this one step further, as components can react to changes in local data, without knowledge of which other components originally performed the change. For instance, our coordination component may generate a new TÆMS structure describing a goal it has agreed to perform. This TÆMS structure is added to the agent's state repository (provided by yet another component), which serves as a common data repository for the components. The scheduling component will react to this addition by producing a schedule for the task structure, which is also

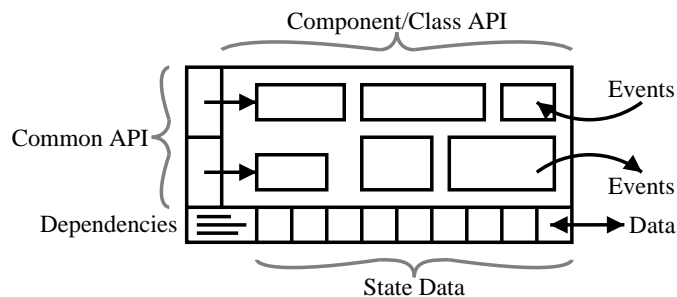


Figure 3: Abstract view of a typical JAF component.

placed in the state repository. This schedule is then used by the execution component to perform the desired actions. Finally, the success of a particular action will cause the coordination component to send back the appropriate results. In each of these steps, the actor or originator could be seamlessly replaced, without the modification of other components in the sequence.

There are several drawbacks to this framework. The most important is the absence of a well-defined thread of control. The control architecture does provide for differentiated execution phases (e.g. construction, initialization, execution), and a periodic execution pulse for each component at runtime. The event system, however, clouds this water considerably, since activity in one component can directly cause a reaction in another. This can lead to long and complicated execution stacks, as well as the potential for cycles or oscillations. Because components are designed by different developers, one cannot predict a priori exactly how they may interact. This process is further complicated by the fact that event distribution is unordered, so one cannot assume that a particular component has processed it before another, and it is difficult to insert new activities between event generation and reaction. A related drawback arises from the inability to preempt a component's execution within the single thread of control. Again, because components may be developed independently, the activity in one may inadvertently cause the failure of another in time- or resource-critical situations. Consider the case above, where coordination generated a new goal for the agent. If the scheduling process takes too long to find an appropriate schedule, or if an unrelated process were to start shortly thereafter, it may become impossible for the deadline agreed upon to be met.

5. CONCLUSION

We have described some of our research in generalized agent technology and pointed out some of the issues we have encountered. Drawing away from the specifics of each component, we would like to leave readers with the thought that developing reusable agent technology is a hard problem. Because understandings evolve, as with all research, technologies must stretch and evolve too. We, of the agent's community, might have a slightly harder problem than researchers in other areas because both the application domains and the agent construction technologies are evolving concurrently. What are the infrastructure requirements of a complex, persistent, autonomous personal assistant that migrates with us from machine to machine, reads our news, filters our mail, and coordinates our activities with peers, family, and friends? We have an idea at this time, but, the landscape is far from being well defined.

6. ACKNOWLEDGEMENTS

We would like to thank the researchers who have continued to help evolve TÆMS and TÆMS based control of software agents, including Victor Lesser, Keith Decker, Bryan Horling, Regis Vincent, Ping Xuan, Shelley XQ. Zhang, Anita Raja, Roger Mailler. Alan Garvey also deserves recognition for his work in Design-to-Time agent scheduling which is the forerunner of DTC. We would also like to acknowledge the efforts of the TripBot team members who have helped to add significantly to the number of universities to which TÆMS control has successfully been ported – they are: John Phelps, Yuhui Qian, Erik Albert, Glene Beane, and Tom Wagner. Also of note, Roy and Elise Turner, also of MaineSAIL, and Tom Wheeler, have made efforts to integrate TÆMS and DTC with their technologies for a digital libraries project.

7. REFERENCES

- [1] Ana L.C. Bazzan, Victor Lesser, and Ping Xuan. Adapting an Organization Design through Domain-Independent Diagnosis. Computer Science Technical Report TR-98-014, University of Massachusetts at Amherst, February 1998.
- [2] Brett Benyo and Victor R. Lesser. Evolving Organizational Designs for Multi-Agent Systems. Department of Computer Science Technical Report TR-1999-00, University of Massachusetts, March 1999.
- [3] F. Christiana. *WORDNET: an electronic lexical database and some of its applications*. 1999.
- [4] K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proceedings of the 1st Intl. Conf. on Autonomous Agents*, pages 404–413, Marina del Rey, February 1997.
- [5] Keith Decker, John Graham, and et al. The decaf agent framework. <http://www.cis.udel.edu/decaf>.
- [6] Keith Decker and Jinjiang Li. Coordinated hospital patient scheduling. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 104–111, 1998.
- [7] Keith S. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.
- [8] Keith S. Decker, Edmund H. Durfee, and Victor R. Lesser. Evaluating research in cooperative distributed problem solving. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Vol. II*, pages 485–519. Pitman Publishing Ltd., 1989. Also COINS Technical Report 88-89, University of Massachusetts, 1988.
- [9] Keith S. Decker and Victor R. Lesser. Coordination assistance for mixed human and computational agent systems. In *Proceedings of Concurrent Engineering 95*,

- pages 337–348, McLean, VA, 1995. Concurrent Technologies Corp. Also available as UMASS CS TR-95-31.
- [10] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36(11):1275–1291, November 1987.
- [11] Regis Vincent et al. Distributed Sensor Network for Real Time Tracking. In *Proceedings of Autonomous Agents (Agents-2001)*, 2001. To appear.
- [12] Bryan Horling. A Reusable Component Architecture for Agent Construction. UMASS Department of Computer Science Technical Report TR-1998-45, October 1998.
- [13] Bryan Horling, Brett Benyo, and Victor Lesser. Using Self-Diagnosis to Adapt Organizational Structures. Computer Science Technical Report TR-99-64, University of Massachusetts at Amherst, November 1999. [<http://mas.cs.umass.edu/bhorling/papers/99-64/>].
- [14] Bryan Horling, Regis Vincent, Roger Mailler, Jiaying Shen, Raphen Becker, Kyle Rawlins, and Victor Lesser. Distributed sensor network for real-time tracking. In *Proceedings of Autonomous Agent 2001*, 2001. To appear.
- [15] N.R. Jennings, J.M. Corera, L. Laresgoiti, E.H. Mamdani, F. Perriollat, P. Skarek, and L.Z. Varga. Using ARCHON to develop real-world dai applications for electricity transportation management and particle accelerator control. *IEEE Expert*, 1995. Special issue on real world applications of DAI systems.
- [16] Victor Lesser, Michael Atighetchi, Bryan Horling, Brett Benyo, Anita Raja, Regis Vincent, Thomas Wagner, Ping Xuan, and Shelley XQ. Zhang. A Multi-Agent System for Intelligent Environment Control. In *Proceedings of the Third International Conference on Autonomous Agents (Agents99)*, 1999.
- [17] Victor Lesser, Keith Decker, Norman Carver, Alan Garvey, Daniel Neiman, Nagendra Prasad, and Thomas Wagner. Evolution of the GPGP Domain-Independent Coordination Framework. Computer Science Technical Report TR-98-05, University of Massachusetts at Amherst, January 1998.
- [18] Victor Lesser, Bryan Horling, and et al. The TÆMS whitepaper / evolving specification. <http://mas.cs.umass.edu/research/taems/white>.
- [19] Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. BIG: A resource-bounded information gathering agent. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, July 1998. See also UMass CS Technical Reports 98-03 and 97-34.
- [20] Victor Lesser, Bryan Horling, Frank Klassner, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. BIG: An agent for resource-bounded information gathering and decision making. *Artificial Intelligence*, 118(1-2):197–244, May 2000. Elsevier Science Publishing.
- [21] Victor Lesser, Bryan Horling, Anita Raja, Thomas Wagner, and Shelley XQ. Zhang. Sophisticated Information Gathering in a Marketplace of Information Providers. *IEEE Internet Computing*, 4(2):49–58, Mar/Apr 2000.
- [22] Victor R. Lesser and Daniel D. Corkill. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):81–96, January 1981.
- [23] Anita Raja, Victor Lesser, and Thomas Wagner. Toward Robust Agent Control in Open Environments. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents2000)*, 2000.
- [24] Regis Vincent, Bryan Horling, Tom Wagner, and Victor Lesser. Survivability simulator for multi-agent adaptive coordination. In *International Conference on Web-Based Modeling and Simulation*, San Diego, CA, 1998. SCS (eds).
- [25] Thomas Wagner, Brett Benyo, Victor Lesser, and Ping Xuan. Investigating Interactions Between Agent Conversations and Agent Control Components. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, Lecture Notes in Artificial Intelligence, pages 314–331. Springer-Verlag, Berlin, 2000.
- [26] Thomas Wagner, Alan Garvey, and Victor Lesser. Complex Goal Criteria and Its Application in Design-to-Criteria Scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 294–301, July 1997. Also available as UMASS CS TR-1997-10.
- [27] Thomas Wagner, Alan Garvey, and Victor Lesser. Criteria-Directed Heuristic Task Scheduling. *International Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also available as UMASS CS TR-97-59.
- [28] Thomas Wagner and Victor Lesser. Relating quantified motivations for organizationally situated agents. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI (Proceedings of ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.
- [29] Thomas Wagner and Victor Lesser. Design-to-Criteria Scheduling: Real-Time Agent Control. In Thomas Wagner and Omer Rana, editors, *To appear in Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, LNCS. Springer-Verlag, 2001. Also appears in the 2000 AAAI Spring Symposium on Real-Time Systems and a version is available as University of Massachusetts Computer Science Technical Report TR-99-58.
- [30] Thomas Wagner, John Phelps, Yuhui Qian, Erik Albert, and Glen Beane. A modified architecture for constructing real-time information gathering agents. In *Proceedings of Agent Oriented Information Systems*, 2001. To appear.
- [31] Shelley Zhang, Anita Raja, Barbara Lerner, Victor Lesser, Leon Osterwiel, and Thomas Wagner. Integrating high-level and detailed agent coordination into a layered architecture. In Thomas Wagner and Omer Rana, editors, *To appear in Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, LNCS. Springer-Verlag, 2001. Abstract also appears in ICMAS-2000.