The Struggle for Reuse: Pros and Cons of Generalization in TÆMS and Its Impact on Technology Transition

Tom Wagner, Bryan Horling, Victor Lesser, John Phelps, and Valerie Guralnik
Honeywell Laboratories and the University of Massachusetts at Amherst
{Tom.Wagner,John.Phelps,Valerie.Guralnik}@honeywell.com and
{bhorling,lesser}@cs.umass.edu

Abstract

TÆMS agents integrate planning, scheduling, and coordination components in a reusable agent construction framework. TÆMS agent components have been used in over a dozen research projects and continue to provide an increasingly refined foundation for rapid construction of multiagent control systems for research and commercial prototype applications. This paper recounts a set of generalization issues that we have encountered in striving to build generic multiagent control components, and how these issues effect the use of TÆMS technology both in research and commercial applications.

Introduction

In this paper we discuss the pros and cons of generalized, reusable, agent control technologies and the experiences gained from employing these technologies in several research projects in academia and industry.

A major thrust of TÆMSrelated work to date is the attempt to create generalized domain independent control components for autonomous intelligent agents. These components include technologies for planning, scheduling, coordination, a framework for agent construction, and a simulation environment. We have been pursuing this line of research for over ten years and the components that we have constructed have been used on more than a dozen research projects. The technology is also currently being used for commercial prototype applications. The motivation behind domain independence is that it enables us to reuse the control components on different applications with few modifications. In the following sections we briefly describe selected components of a TÆMS agent, discuss their strengths and weaknesses, and discuss issues that have arisen during their use across different research projects.

The target of our reuse concerns are the hard control problems associated with agents like the BIG Information Gathering Agent [13] and [9; 3], where agents are situated in an environment, able to sense and effect, and are required to solve control problems with many alternative courses of action, and where the outcomes of any one action are not certain before runtime. The way that uncertainty in control is accounted for in TÆMS is through statistically characterized performance characteristics and relationships between the tasks.

The approach to domain independence in TÆMS is achieved by abstracting away 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 [5; 12]. TÆMS task structures are hierarchical task networks, with additional graph structures called *interrelationships*. Notable features of TÆMS models include the explicit representation of task alternatives, the representation and quantification of task interactions, and the characterization of primitive actions in term of quality, cost, duration outcome probability distributions. A sample TÆMS task structure is shown in Figure 1. Task structures are generated from domain theories; domain theories encode ways from moving the agent from one worldstate to another, ideally moving the agent closer to one or more pending goals. In some cases, the domain theories are encoded in a programming language. In the Tripbot domain, we begun work on a domain-independent component to assess an agent's candidate plans in the context of a set of goals[22]. The Design-To-Schedule (DTS) planner module accepts domain theories encoded in a planning language, and, given a problem specification, computation duration limit, and a TÆMS analysis criteria definition, generates TÆMS task structures that solve the goals across method performance possibilities through a set of plans.

Once an agent's alternatives are encoded in TÆMS, the Design-to-Criteria scheduler creates a set of schedules which are the result of criteria-directed decisions about which tasks an agent should execute, what resources it should use, which tasks it should coordinate over with other agents, and how the agent can meet real-time deadlines and resource constraints [21; 18; 17; 15].

When multi-agent systems are constructed, a coordination module, such as GPGP [6; 11] or one of its descendants [19], or a more custom component [10; 16], is added to the agent. The architecture of a typical TÆMS agent is shown in Figure 2.

The problem solver can also be simulated. This is the case in the aircraft readiness application in which agents coordinate simulated aircraft service teams. Because the agents do not actually repair the aircraft, the methods are simulated and TÆMS models of the repair processes are generated offline.

In the repair application, aircraft land at unpredicted times and are assessed for repair and service needs. The aircraft have deadlines by which they must be ready for their next mission and different aircraft have different deadlines. TÆMS agents coordinate the distributed scheduling of the repair teams to ensure that the maximum number of aircraft are serviced and ready to launch by their respective deadlines. The problem is made difficult by the interactions between the activities of different service teams, e.g., the engines cannot be serviced while missiles are being reloaded. Control in this application is decomposed into two types of activities: 1) communication based coordination and 2) local agent scheduling. As with most TÆMS applications the coordination technologies handle the communication of partial task structures and the formation of commitments between agents about which agent will do what task and consume which resources and when. These commitments are then passed to the local DTC scheduler along with the agent's other problem solving options and its current objective function, e.g., maximize solution quality or trade-off solution time with resource consumptions, and the scheduler decided on a course of action for the agent.

TÆMS agents can be constructed incrementally. A domain problem solver with TÆMS and the DTC scheduler is an agent capable of meeting real-time deadlines and performing performance tradeoffs. When a GPGP module is added, the agent can interact with other agents to perform cooperative or joint problem solving. Other modules can be added to support organizational control, learning,

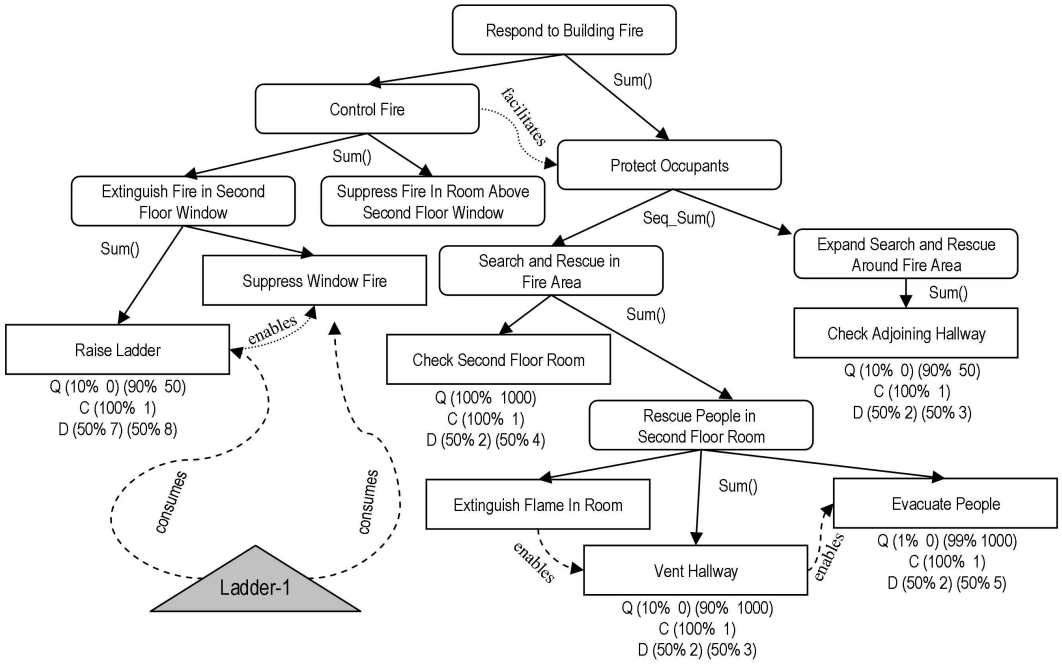

Figure 1: Simplified Civilian Emergency First-Responder Task Structure

and/or other styles of coordination. This domain-independence coupled with a component 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 [14; 13], the Intelligent Home project (IHome) [10], the DARPA ANTS real-time agent sensor network for vehicle tracking [16], distributed hospital patient scheduling [5], distributed collaborative design [7], process control [23], the TripBot [22], agent diagnosis [1], and others. However, in many of these projects modifications to the artifacts were required and we became aware of certain design decisions that affected the ability to move to a new application, change the control flow within the agent, or to expand the TÆMS task modelling language.

The Java Agent Framework (JAF) [8] is the agent component framework, or ”glue” by which the domain-independent and domain-dependent components are integrated on projects like the IHome [10] and the ANTS real-time agent sensor network for vehicle tracking [16]. Like Decker’s DECAF [4], 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 that enables different JAF agents to “execute” TÆMS primitive actions in a controlled environment.[1]

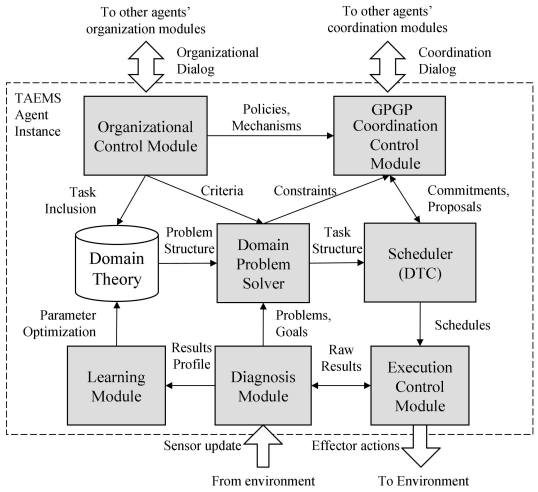

Figure 2: TÆMS Agent Architecture

TÆMS Task Models

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 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 modelled 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.

[1] 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.

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 1, which is a conceptual, simplified sub-graph of a task structure used in a civilian first-responder application that we are currently developing. It describes a portion of the building fire response process. The top-level task is to *Respond to Building Fire*. It has two subtasks, *Control Fire* and *Protect Occupants*, both of which are decomposed into subtasks or methods. The methods are that are statistically described in terms of their expected quality, cost, and duration. The *enables* arc between *Raise Ladder* and *Suppress Window Fire* is a non-local-effect (nle) or task interaction; it models the fact that the act of suppressing a given window fire requires raising a fire ladder truck's ladder. Other task interactions modelled in TÆMS include: *facilitation*, *hindering*, *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, *Protect Occupants* has two subtasks, joined under the *seq_sum()* quality-accumulation-function (*qaf*), which defines how performing the subtasks relate to performing the parent task. In this case, both methods must be performed in order for there to be utility in the parent task, and where the parent task's utility if this constraint is not violated is the sum of its subtasks. The *sum* qaf governs the *Search and Rescue in Fire Area* task, indicating that either method in either order may be executed to generate utility. In general, a TÆMS task structure represents a family of plans and schedules, rather than a single plan or schedule, where the different paths through the network exhibit different statistical characteristics that must be compared to find the best alternative.

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 with or 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 [12], 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 the DTC scheduler and JAF, the existence of tutorial style documentation helped significantly.

The development a of usable and sharable TÆMS modelling 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 the DTC scheduler 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. The DTC scheduler 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 modelling 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 de-

[2] Facilitation 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.

pendency specification used in the TripBot [22]. 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 [20]). However, most of these issues fall into category of basic research questions.

Design-to-Criteria Scheduling

The Design-to-Criteria (DTC) scheduler is the agent's local expert on making scheduling 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 modelled 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 the DTC scheduler is designed for open environments where unpredicted change is commonplace. When change occurs, the agent reinvokes the DTC scheduler 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. The DTC scheduler is the agent's scheduling trade-off and control expert. The TÆMS scheduling objective is not to sequence a set of unordered actions but to *find* within a generated task structure a set and then a *sequence* of actions that *best* suits a the agent's current objectives; that is to say that there may be mutually exclusive sets of partially ordered sets of actions that must be sequenced.

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, heuristic decision making, and heuristic error correction. The algorithm and techniques are documented more fully in [21; 18; 17; 15].

The Good, the Bad, and the Ugly

The DTC scheduler 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 8 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 the DTC scheduler running in exhaustive scheduling mode will run in less time than if running in its normal heuristic mode which is designed to cope with high order combinatorics from many different sources. In some cases, for some task structures, doing an exhaustive search 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 the DTC scheduler. The thought here is that domain independence in control is a hard problem and there are always performance trade-offs involved. Striving for generality means to have to address the hardest class of problems possible for a given problem instance.

The DTC scheduler was also 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 extensible 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 the DTC scheduler has been its encapsulation. The DTC scheduler 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 the DTC scheduler, 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 concerns has paid great dividends. Evidence of this includes the variety of ways that the DTC scheduler has been used in our research, including other scheduler enhancements that build on top of the scheduler as an external clients, e.g, work in schedule caching and partial-ordering [16].

Related to the DTC scheduler's encapsulation was the construction of a human readable textual I/O format for DTC. Referred to as *t-TÆMS* , the input to the DTC scheduler 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, the scheduler schedules and produces multiple different output files. One of which is a t-TÆMS schedule file that contains a ranked set of detailed schedules for the agent that includes 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 replan or reschedule. The DTC scheduler also produces a more human friendly schedule representation and a simple schedule description file for clients that do not which 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 the DTC scheduler. While the scheduler 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, the DTC scheduler does not require the client to specify anything him or herself. Instead it is configured with a set of default options that, in general, yield good performance.

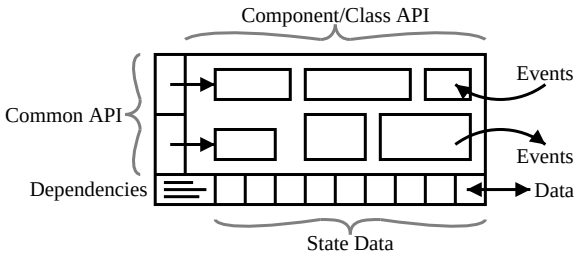

Figure 3: Abstract view of a typical JAF component.

It is worth noting that it an ideal world it 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 in developing the DTC scheduler. Reuse requires support to evolve reusable artifacts. 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 of infrastructure sharing within a lab should also be recognized and explicitly addressed.

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 , 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 , 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 deployed in a distributed sensor network environment 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 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, how-

ever, 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.

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.

Acknowledgments

We would like to thank the researchers who have continued to help evolve TÆMS and TÆMS based control of software agents, including Keith Decker, Alan Garvey, Bryan Horling, Regis Vincent, Ping Xuan, Shelley XQ. Zhang, Anita Raja, Roger Mailler, and Norman Carver. We would also like to acknowledge the efforts of the many other contributors on TÆMS related projects and those in the larger community who have helped to shape these ideas.

References

[1] A. Bazzan, V. Lesser, and P. Xuan. Adapting an Organization Design through Domain-Independent Diagnosis. CS Tech. Report TR-98-014, UMASS, 1998.

[2] B. Benyo and V. Lesser. Evolving Organizational Designs for Multi-Agent Systems. CS Tech. Report TR-1999-00, UMASS, 1999.

[3] K. Decker, A. Pannu, K. Sycara, and M. Williamson. Designing behaviors for information agents. In *Proc. of the 1st Intl. Conf. on Autonomous Agents*, February 1997.

[4] K. Decker, J. Graham, et al. The decaf agent framework. http://www.cis.udel.edu/ decaf.

[5] K. Decker and J. Li. Coordinated hospital patient scheduling. In *Proc. of the Third Intl. Conf. on Multi-Agent Systems (ICMAS98)*, 1998.

[6] K. Decker. *Environment Centered Analysis and Design of Coordination Mechanisms*. PhD thesis, UMASS, 1995.

[7] K. Decker and V. Lesser. Coordination assistance for mixed human and computational agent systems. In *Proc. of Concurrent Engineering 95*, 1995. Also avail. as UMASS CS TR-95-31.

[8] B. Horling. A Reusable Component Architecture for Agent Construction. CS Tech. Report TR-1998-45, UMASS, 1998.

[9] N.R. Jennings et al Using ARCHON to develop real-world dai applications for electricity transportation management and particle accelerator control. *IEEE Expert*, 1995.

[10] V. Lesser et al A Multi-Agent System for Intelligent Environment Control. In *Proc. of the Third Intl. Conf. on Autonomous Agents (Agents99)*, 1999.

[11] V. Lesser, K. Decker, T. Wagner et al Evolution of the GPGP Domain-Independent Coordination Framework. To appear in the Journal of Autonomous Agents and Multi-Agent Systems in 2003.

[12] V. Lesser, B. Horling et al. The TÆMS whitepaper / evolving specification. http://mas.cs.umass.edu/research/taems/white.

[13] V. Lesser et al BIG: An agent for resource-bounded information gathering and decision making. *Artificial Intelligence*, 118(1-2):197–244, May 2000. Elsevier.

[14] V. Lesser et al Sophisticated Information Gathering in a Marketplace of Information Providers. *IEEE Internet Computing*, 4(2):49–58, Mar/Apr 2000.

[15] A. Raja, V. Lesser, and T. Wagner. Toward Robust Agent Control in Open Environments. In *Proc. of the Fourth Intl. Conf. on Autonomous Agents (Agents2000)*, 2000.

[16] B. Horling, R. Mailler, J. Shen, R. Vincent, and V. Lesser. Using autonomy, organizational design and negotiation in a distributed sensor network. Book chapter. To appear 2003.

[17] T. Wagner, A. Garvey, and V. Lesser. Complex Goal Criteria and Its Application in Design-to-Criteria Scheduling. In *Proc. of the 14th National Conf. on Artificial Intelligence*, 1997. Also available as UMASS CS TR-1997-10.

[18] T. Wagner, A. Garvey, and V. Lesser. Criteria-Directed Heuristic Task Scheduling. *Intl. Journal of Approximate Reasoning, Special Issue on Scheduling*, 19(1-2):91–118, 1998. A version also avail as UMASS CS TR-97-59.

[19] T. Wagner, V. Guralnik, and J. Phelps. Software agents: Enabling dynamic supply chain management for a build to order product line. To appear in the Journal of Electronic Commerce Research and Applications, Elsevier, 2003. A version avail at http://www.drtomwagner.com

[20] T. Wagner and V. Lesser. Relating quantified motivations for organizationally situated agents. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI (Proc. of ATAL-99)*, Springer-Verlag, 2000.

[21] T. Wagner and V. Lesser. Design-to-Criteria Scheduling: Real-Time Agent Control. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, LNCS. Springer-Verlag, 2001.

[22] T. Wagner, J. Phelps, Y. Qian et al A modified architecture for constructing real-time information gathering agents. In *Proc. of Agent Oriented Information Systems*, 2001.

[23] S. Zhang et al Integrating high-level and detailed agent coordination into a layered architecture. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, LNCS. Springer-Verlag, 2001.

[24] R. Vincent, B. Horling, T. Wagner, and V. Lesser. Survivability simulator for multi-agent adaptive coordination. In *Intl. Conf. on Web-Based Modeling and Simulation*, 1998.