Automating Human Based Negotiation Processes for Autonomic Logistics[1]

Gabor Karsai
Institute for Software-Integrated Systems
Vanderbilt University
P.O.Box 1829B
Nashville, TN 37235,USA
615-343-7471
gabor@vuse.vanderbilt.edu

George Bloor
Phantom Works
The Boeing Company
(206)655-9479
George.Bloor@jsf.boeing.com

Jon Doyle
Laboratory for Computer Science
Massachusetts Institute of Technology
545 Technology Square, Room 419
Cambridge, Massachusetts 02139-3539, USA
(617) 253-3512
doyle@mit.edu

Abstract— Affordability and responsiveness are two key requirements for the next generation of aircraft support systems. Aircraft have to be repaired with minimum downtime and the support process has to be economical and efficient. The vision of Autonomic Logistics (AL) entails a maintenance and support system that can autonomously respond to "events", e.g. problems detected on-board the aircraft. The response includes identifying the source of the problem, acquiring the correct parts and tools, locating and scheduling the right maintenance personnel. Currently, software tools and packages are available that can provide functional components for an AL system, but the integration of these does not exist yet.

The paper discusses how agent technology in general and negotiation processes in particular can be applied to build an AL system. We envision a system where existing, "legacy" packages are integrated via an agent framework: a higher-level layer that ties together the packages. Agents represent tasks that need to be performed in an AL system, and they autonomously exist, navigate, and negotiate within the framework. This approach, while emphasizes automation, is not that different from existing, human processes. Even in existing maintenance organizations human negotiation is an accepted practice, which, in many cases, "makes the system work", in spite of rigid, formal rules. Thus, we envision that the autonomous, negotiation-based system will provide the required capabilities for an AL system to increase readiness and turn-around rates, and to decrease costs.

TABLE OF CONTENTS

1. INTRODUCTION

Affordability and responsiveness are two key requirements for the next generation of aircraft support systems. Aircraft have to be repaired with minimum downtime and the support process has to be economical and efficient. Today's information systems that support the maintenance process are typically not able to satisfy these requirements. Isolated, highly sophisticated, single-purpose systems are commonplace, which do an excellent job in their own specialty, but can rarely communicate with each other. The pervasive problem of these systems is the lack of semantic interoperability among them.

In order to enhance the support process, one has to consider the efficient integration of various subsystems. The

[1] 0-7803-5846-5/00/$10.00 © 2000 IEEE

pilot reports problems with the aircraft using a pilot debriefing system. This should lead to diagnostic analysis, possibly done by a sophisticated ground based diagnostic analyzer system that also has access to on-board flight data. The result of the diagnostic analysis is a set of maintenance tasks including both tests and repair actions, that need to be executed. The maintenance tasks require skilled mechanics (certified to do the specific action) from the pool of available mechanics, and Line Replaceable Units (LRU-s) that need to be replaced. The assignment of mechanics to tasks is a resource allocation problem, similarly to the part provisioning. Once the mechanic fixed and tested the aircraft, a log of this work has to be created with all the accompanying documentation (e.g. serial numbers of parts replaced, etc.). In order to collect statistics for the military, this data has to be relayed to a maintenance information database. In order to provide feedback to the weapon system supplier the failure and maintenance data should be fed back into continuous engineering/improvement process.

For each step of the above process information system components do exist, but their integration is lacking. What is also lacking is the *autonomy* of the process: data is shipped with human intervention, and the process is not very automatic (in spite of the fact that most of the data exist in electronic form).

Another interesting observation is that the process relies heavily on human interaction for conflict resolution. Resource conflicts arise all the time, but humans have very good skills to resolve them. The process providing the resolution is typically a *negotiation process* where several parties gradually arrive at a mutually acceptable solution. Another method of conflict resolution relies on taking the issue up to the level of hierarchy with the first common higher authority. This *hierarchical conflict resolution* is typical in organizations with a strict chain of command (like the military). It is important to recognize that the conglomerate of the information systems used in support operations is currently not capable of supporting these conflict resolution strategies.

In this paper we discuss a vision for a new generation of logistic/maintenance support systems, and how it may be supported using agent technology in conjunction with existing, legacy systems. Specifically, we present some ideas how complex resource allocation processes (currently done by humans) can be automated by mapping them into agent negotiation strategies.

2. BACKGROUND

The Autonomic Logistics vision. The Autonomic Logistics (AL) [1] concept envisions a maintenance and support system that can autonomously respond to "events", e.g. problems detected on-board the aircraft. Thus, instead of the current, "manual" processes, the AL system employs a computational infrastructure that makes the autonomic response possible. The response may include various actions:

- identifying the source of the problem,
- acquiring the correct parts and tools,
- locating and scheduling the right maintenance personnel,
- logging the maintenance task when finished.

Currently, software packages, systems and databases are available that can provide the functional components performing these actions for an AL system, but their integration is not a solved problem.

The AL system is a step toward supporting condition-based maintenance of aircraft. For condition-based maintenance one needs the continuous monitoring of the health of the aircraft (and all of its subsystems), and in the case of an impending failure a maintenance action must be scheduled to prevent further degradation or total failure. Non-autonomic, manual maintenance processes may not be fast and flexible enough to handle this type of maintenance scheduling and resource allocation problems.

To summarize, an Autonomic Logistic system can be defined as a maintenance and supply system wherein change in the health of aircraft triggers the logistics system to

- identify, locate, gather, and schedule parts, equipment, and technical personnel,
- maintain stocks,
- perform data analysis and provide feedback to manufacturers,
- resolve conflicts and allocate scarce resources.

It is important to emphasize that large and sophisticated information system components do exist that are capable of solving these problems independently. However, to realize an integrated AL system, the integration of these components is required, and in such a manner that resulting "super-system" exhibits autonomic behavior, with no or minimal human intervention.

Another aspect of current processes and systems is (which is not related to computing) that often human goals and preferences do get involved in the process. In the case of resource conflicts, frequently humans resolve the issues via communication, interactions, and negotiation. Also, parts distribution, resource scheduling, personnel assignments are oftentimes negotiated. This "intervention" gives the system its flexibility, and our goal is to preserve this property even in a fully autonomic system

Agent technology. Agent technology is an outgrowth of distributed AI [2], which uses active, typically mobile components to facilitate problem-solving processes. While there seems to be no single, formal definition for an agent, in our work we found it practical to consider them as sophisticated objects that are dynamically created, and through communication, cooperation, and competition solve complex problems. Agents typically represent the goals of stakeholders external to the system proper, on whose behalf they act.

Recently, agent technology became widely available with arrival of the Java language that supports network computing at various levels. Beyond the basic needs of agents (communication), the Java run-time system makes possible agent migration across nodes of a network, dynamic

service lookup on a subnet (using the Jini technology [3]), and distributed blackboard-style communication models (using the JavaSpaces technology [4]). Although there are other techniques available for agent implementation (e.g. AgentTcl, Lisp, or C++), Java seems to be language of choice.

Agent technology as an attempt to simulate the real world has been successfully applied in many laboratory situations. Agent technology as a novel way of building and integrating systems from components has been applied in a number of cases, but mostly for small-scale, research-oriented environments. Agent-based systems for large-scale Agent technology as a practical approach for engineering large-scale distributed systems is still in its developing phase. There are a number of agent-building toolkits available, but building systems using agents is not a well-established discipline.

In many agent-oriented solutions, the agents must compete for resources to perform their tasks. This competition has to be resolved, and one relevant technique is that of negotiation. Problem solving through negotiation is a concept that was introduced earlier in the field of distributed AI. An early result, the Contract Net [5] protocol, demonstrates how task distribution can be arranged among agents. Another important application of negotiation is related to resource allocation problems. Negotiation provides an iterative method for arriving at the solution that is mutually acceptable to all participants. Note that this solution is not necessarily the "best" or "optimal" solution, but rather "acceptable for all".

Model-Integrated Computing (MIC). MIC is a system development approach [6] that is based the extensive use of models in the development process. The use of models in itself is not a new idea. Various analysis and design techniques (especially the object-oriented ones) very frequently build models of the system before realization, and model its environment as well. However, we have extended and specialized the modeling process so that the models can be more tightly integrated into the system development cycle than in traditional techniques. The process supporting this activity is called Model Integrated Computing (MIC), and it results in a model-integrated system (MIS). In an MIC process the models describe the environment of the system, represent the system's architecture and they are used in generating and configuring the system. These models are indeed integrated with the system, in the sense that they are active participants in the development process, as opposed to being mere passive documents. When MIC is used in developing a system, models are involved in all stages of the life cycle. To support this, the initial step is the building of tools that support model creation and editing, used by the end-users when they want to customize the final application. The model-editing tools are typically graphical, but more importantly, they support modeling in terms of the actual application domain. This domain-specific modeling is essential for making end user programmability feasible. There are two interrelated processes in MIC: (1) the process that involves the development of the model-integrated system, and (2) the process that is performed by the end user of the system (in order to maintain, upgrade, reconfigure) the system -- in accordance with the changes in its environment. Figure 1 shows the processes schematically. The first process is performed entirely by the system's developers (i.e., software engineers), the second, usually, by the end users. To summarize, in MIC the system is created through the development of the following: (1) a modeling paradigm, (2) the model builder (editor) environment, (3) the model interpreters and (4) the runtime support system. The product of this process is a set of tools: the model builder, model interpreter and generic runtime support system. Using these, first the developers, but eventually the end users can build up the application itself by going through the following steps: (1) develop models, (2) interpret the models and generate the system (this step is automatic), and (3) execute the system. The key aspect of the development process is that domain specific models are used in building the application, and thus it can be re-generated by the end-users. It may seem that MIC necessitates a bigger effort than straightforward application development. This is true if there is no reuse and every project has to start from scratch. In recent years we have developed a toolset called the Multigraph Architecture (MGA)[7] that provides a highly reusable set of generic tools to support MIC. We claim that the tools provide a *meta-architecture*, because instead of enforcing one particular architectural style for development, they can be customized to create systems of widely different styles. Figure 1 shows the components found in a typical MGA application. The shadowed boxes indicate components that are generic and are customized for a particular domain.

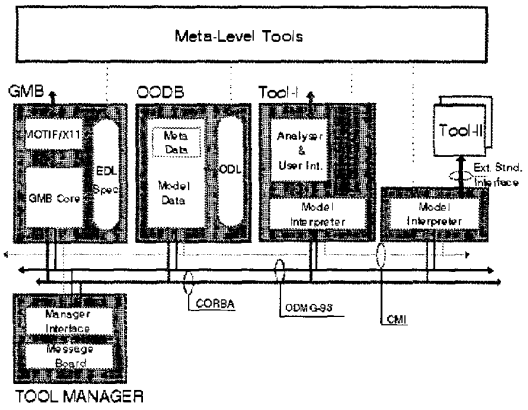

Figure 1: Multigraph Architecture

3. AGENT-BASED NEGOTIATION APPROACH

We envision the use of *agent technology* in general, and *negotiation technology* in particular, and *model-integrated computing* to build an autonomic logistics system. We consider the AL problem as building a new type of complex information system, which heavily relies on existing, legacy

systems, and exhibits autonomic behavior. The above three technologies do offer the right ingredients for constructing such a system, as we show it below.

Agents and autonomous negotiating teams

Traditional, legacy systems used in maintenance logistics are "passive". Users interact with them, and the systems react to user commands, but they very rarely initiate actions on their own. In addition, legacy systems provide high quality, "point solutions" for specific functions required in an information system architecture.

The AL vision requires an affordable and seamless integration of these systems that will result in a "super-system" that exhibits the desired autonomic behavior. The heavy modification of these systems is not feasible: it simply takes too much effort to change them. Tight integration (with strict synchronization of the systems, mutual interdependence, etc), is probably not a good idea either: it makes the architecture too rigid. We need a relaxed integration mechanism that still provides the functionality desired, but does not impose strict constraints on the architecture.

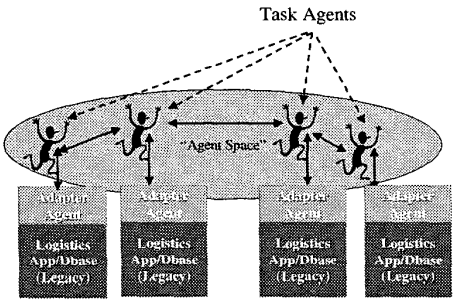

Figure 2: Agent-based Architecture

Based on these observations, we recommend the use of an agent-based approach for realizing the AL vision. Figure 2 shows a high-level representation of the proposed architecture. In this system, agents act as representatives of (1) component systems and (2) tasks to be performed. The two agent types are called *adaptor agents* and *task agents*. Adaptor agents represent external packages and they act as interface between task agents and the legacy systems. Their functionality is rather limited: managing the interaction with a legacy system. Task agents represent the tasks to be performed. What precisely these tasks are is dependent on the application domain, Autonomic Logistics in our case.

On a high-level, the system operates as follows. When a legacy system causes a "significant event" its adaptor agent will notice it. The adaptor agent then creates a specific type of task agent that is responsible for performing the related task. The adaptor agent is activated, and it creates other task agents as required. Either the task itself, or its subservient task agents communicate with the adaptor agents of other legacy systems in order to perform their task. When a task agent has accomplished its task, it reports to its supervisor, and then terminates. Obviously, the specific workings of the system are determined by the actual operational scenarios ("use-cases") of the AL domain.

The task agents, when operating, will have to acquire resources. During this process they will get into conflicts, and conflict resolution is necessary. The envisioned solution for conflict resolution is to use negotiation protocols. We envision negotiation as a process that is based on the active information exchange between task agents, and which leads to a mutually agreeable solution through an iterative process.

There are several reasons why the negotiation is necessary in the system.

1. Resource allocation and planning solutions must be distributed. Maintenance logistics systems, especially for the military, are inherently distributed. Resources, agents, and communication facilities are necessarily distributed. The coordination of these distributed components is a significant problem, because only incomplete information is available locally, thus participating systems have only incomplete knowledge of the global situation. Organizational preferences change independently, and changes are not propagated immediately, if at all. The AL system should take into consideration all needs from units, but also supply, transport and organization capabilities. To summarize: it is simply infeasible to centralize all knowledge required for the solution.

2. Resource allocation and planning solutions must adapt to changing circumstances. Frequently-changing circumstances undercut fixed designs. If no negotiation is applied, instead of arriving at a mutually acceptable solution through an iterative process, one has to resort to inflexible, rigid rules. Communication links or even organizational structures may change, perhaps even fail. A negotiation-based approach is much more amenable to address these problems, than a fixed, static design. In many cases, participants are not known in advance, often they are discovered dynamically. Because responses from newly discovered participants cannot be calculated in advance, the iterative, negotiated approach is more suitable. For example, for allocating air squadron maintenance personnel, tools, and parts, the location of desired assets and the availability of team members with needed skills has to be determined through a highly dynamic process.

3. Frequently, to arrive at a solution one has to make tradeoffs between values. For instance, preferences regarding the speed of getting a result vs. the risk it involves may have to be decided dynamically. Additional complexity results when desired tradeoffs change with the situation. A negotiation approach can easily tolerate these changes.

4. Resource agreements must assure user confidence. Results that are imposed externally, say by organizational rules, are often considered unacceptable. Results that are derived based on local preferences are deemed to be more acceptable. Negotiation rules and preferences that clearly express the local goals of an organization facilitate solutions that are easier to understand, justify, and accept by the

members of the organization.

We envision negotiation as the tool for facilitating conflict resolution in case of resource conflicts. Some examples are listed below.

1. Dynamically generated maintenance tasks require mechanics with specific skills. When tasks conflict over allocating the "resources" (i.e., the mechanics), the agents representing the tasks can arrive at mutually acceptable solutions through a negotiation process.
2. Maintenance schedule items (e.g. `<mechanic, part set, start time, duration>` tuples) need to be modified such that overall performance is increased. One performance measure may be the sortie generation the unit is capable of. The task agents that "caused" the maintenance tasks may have to negotiate with each other to perform the re-scheduling. In this case negotiation will result in a distributed scheduling algorithm.
3. Maintenance actions may have to be deferred because of conflicts with the operational schedule. The resource is the aircraft, on which maintenance tasks need to be performed but it also has to take part in operations. To resolve these conflicts, the task agents representing the desired maintenance actions and the task agents representing the operational requirements need to come to a mutual agreement.

We envision that negotiation illustrates intelligent action in open, dynamic environments. The negotiation technology applies to a large number of resource allocation problems, including scheduling physical, computational, organizational, and planning resources. However, this is not to say that some problem domains do not require other approaches. For instance, fixed problems and extremely small time constants may admit better, special solutions. It is also interesting that negotiation may also be applicable to specification problems, for instance mission statements, task specifications, etc.

System integration issues

The MIC/negotiation approach uses the already existing, legacy systems. This is a requirement, because rewriting these systems is not feasible: it is simply too costly. The Adaptor agents of the architecture are responsible for interfacing these systems to the rest of the agent-based infrastructure. We call this new part of the system the "Agent Space" where the agents are created and function. The agent space provides the infrastructure for basic agent services: life-cycle, communication and interaction, and coordination. Naturally, this infrastructure should be flexible enough to support all varieties of systems, agent interaction protocols, etc.

Building an AL information system gives new challenges for system integration. Some of the requirements are listed below.

1. Legacy systems may change over time. The applications that are integrated in this framework change over time. New capabilities can be added to them, and occasionally they can be replaced by newer systems. These changes should have minimal impact on the AL system as a whole.
2. Legacy systems may be relocated on the net. When military operations require, entire maintenance systems should change their geographical location, and thus their "place" on the network as well. These changes should not impact the AL system.
3. Sophisticated users should be able to fine-tune the AL system to represent better the goals the preferences of their organizations. These changes should be seamlessly integrated in the system.
4. Task Agents (service clients) should be able to locate Adaptor Agents (service providers) without knowing their locations and addresses in advance. Legacy systems may come and go, and the task agents should be able to locate them without problem.

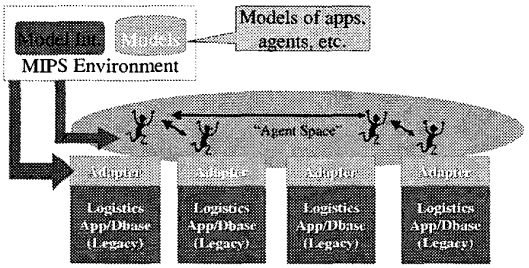

Figure 3: Model-Integrated Program Synthesis and Agents

Using existing technology can solve some of this integration issues. For instance, 2.) and 4.) are potential candidates for using Jini technology for service location. The idea is that the adaptor agents register themselves with a directory (a kind of "trader") service. The task agents use Jini's broadcast facilities to locate the directory service objects first, then they execute a search on them in order to locate the service provider/adaptor agent. While Jini provides the basic infrastructure, probably higher performance lookup capabilities are needed. Jini is using simple string-lookup, but a unification-based approach may be required.

For the other system integration issues we envision the use of Model-Integrated Computing (MIC). Figure 3 shows how MIC can play a role in building an AL system. Specifically, the modeling and synthesis capabilities provided by MIC can be utilized as follows.

1. Model-based generation of task agents. A suitable modeling paradigm (language) can be used to capture the interaction protocols, goals, preferences, behavior, etc. of agents. These models can be specified in terms of abstract diagrams that can be used for synthesizing the code that implements the agent behavior. For instance, interaction protocols can be described using

diagrams that represent communicating finite state machines (CFSM-s). The diagrams can then either be "interpreted" at agent run-time, or compiled into efficient code for execution.

2. Model-based generation of adaptor agents. The adaptor agents act as "translators" that transform agent messages into operations on the legacy system, and events detected in the legacy system into messages for the other agents. This on-line translation of messages can also be modeled. There are various techniques one can employ, CFSM-s being one example.

The model-based approach helps in evolving the system because the behaviors are encoded in high-level, abstract structures. When a legacy system changes, the model used to generate its adaptor agent has to be changed, and regenerated. When a new behavior for a task agent is desired, its behavioral model should be changed and its code regenerated.

An additional benefit is that models lend themselves better to verification than straight code. For instance, when an interaction protocol is implemented in software, it can be very hard to verify. If it is expressed in terms of an abstract mathematical structure, verification is easier. Obviously, the approach will result in an acceptable solution only if the generation/synthesis process that transforms the models into code is correct.

4. Conclusions

In this paper we have shown an initial design for building an Autonomic Logistic system from already existing and new components. The use of agent technology makes possible the integration of these components in such a manner that new functionalities (including autonomy) can be provided easily. The use of modeling and model-based generation of agents facilitates end-user programmability and evolvability.

We are now working on a small-scale prototype of the above system that demonstrates the basic capabilities. A significant effort should go into developing interaction protocols and mechanisms. Published negotiation techniques do not always deal with real-life situations (e.g., broken communication links, server problems, etc.). We plan to develop the robust interaction protocols to address these issues.

Representing goals and preferences using abstract models is also an open problem. We are working on developing the modeling language that allows us to capture these. Obviously, their translation into efficient agent code is an additional problem.

To summarize, agent-based integration solutions seem to offer good capabilities for realizing the Autonomic Logistic vision. Model-integrated Computing techniques can provide the framework for implementing an agent-based solution that is flexible, extensible, and evolvable.

5. Acknowledgements

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-050. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

[1]URL: http://www.jast.mil/html/phm.htm

[2]Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 1999.

[3]URL: http://www.sun.com/jini

[4]URL: http://java.sun.com/products/javaspaces

[5]Smith, R. "The Contract Net Protocol: A High Level Negotiation Protocol for Distributed Problem Solving," IEEE Transactions on Computers (29) 1980.

[6]Misra A., Karsai G., Sztipanovits J.: "Model Integrated Development of Complex Applications", Proceedings of the Fifth International Symposium on Assessment of Software Tools, Pittsburgh, PA, June, 1997.

[7]Sztipanovits J., Karsai G., Franke H.: "Model-Integrated Program Synthesis Environment", Proceedings of the IEEE Symposium on Engineering of Computer Based Systems, Friedrichshafen, Germany, March 11-15, 1996

Gabor Karsai *is Associate Professor of Electrical and Computer Engineering at Vanderbilt University and co-director of the Institute for Integrated Information Systems. He has over twelve years of experience in software engineering. He conducts research in the design and implementation of advanced software systems for real-time, intelligent control systems, and in programming tools for building visual programming environments, and in the theory and practice of model-integrated computing. He received his BSc and MSc from the Technical University of Budapest, in 1982 and 1984, respectively, and his PhD from Vanderbilt University in 1988, all in electrical and computer engineering. He has published over 60 papers, and he is the co-author of four patents.*

George Bloor *is a Senior Principal Engineer working for the Boeing Joint Strike Fighter Program and is currently serving as the lead engineer for the Joint Strike Fighter's Prognostic and Health Management Test Bench. He joined Boeing's Advanced Research Group, then known as the High Technology Center, in 1987. While at Boeing, he has worked in the disciplines of telecommunications, flight controls and avionics. Prior to joining Boeing, George held positions at Hewlett-Packard and at AT&T Bell Labs. He has earned a Masters Degree*

in Electrical Engineering from the University of Washington and a Masters Degree in Mathematical Statistic from Iowa State University. George has published materials in the IEEE, SMC, and for several other professional societies.

Jon Doyle *is Principal Research Scientist at the Laboratory for Computer Science at MIT. His principal research goal is to develop theories and techniques for representation and reasoning that have a sound basis in decision theory, economics, and logic, and to apply these to practical problems of monitoring and medical informatics, especially to the representation and use of qualitative and quantitative models of preferences and utilities. He has made many contributions to the theory of rational and economic reasoning, has developed representations for rational agents and market-guided reasoning systems, and has worked on structuring ontologies for planning, command and control, and general medical health-maintenance monitoring. He is a Fellow of the American Association for Artificial Intelligence, a former AAAI Councilor, and a former president of ACM SIGART.*