When I want to go out to the movies, rather than read reviews, I ask my sister-in-law. We all have an equivalent who is both an expert on movies and on expert on us. What we need to build is a digital sister-in-law.

On “Agents” by Nicholas Negroponte, Media Lab, M I T
Agents

Where Artificial Intelligence meets Natural Stupidity

Summary Prepared by Dr Shoumen Datta
Engineering Systems Division, School of Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139

Disclaimer

This summary on Agents is not an original work by the author (Shoumen Datta). I have merely written it in simple English to make it suitable for understanding by a large number of non-experts, myself included. I neither have the talent nor the training to produce the concepts and formulations described in this article. Failure to represent the ideas with clarity is entirely my fault. If you have understood and appreciated the scope of Agents, then, the credit is solely due to the brilliance of the scientists whose work I have quoted/paraphrased. I have used papers from Massachusetts Institute of Technology (MIT), Carnegie-Mellon University (CMU) and University of Michigan, Ann Arbor. The monographs by H. Van Dyke Parunak deserves special acknowledgement. Email: shoumen@mit.edu
Abstract
Linearisation of real world conditions to fit mathematical models may create lack of real-time adaptability in supply chain. An example of this is the Bull Whip Effect that depicts wide fluctuations in supply chain. The discrete, dynamic and distributed nature of data and applications require that supply chain solutions not merely respond to requests for information but intelligently anticipate, adapt and actively support users. Agents can support clearly discernible tasks or process, interact with each other in a specified environment (say, inventory management), interact with other Agents directly or via a message bus, continuously harness real-time data (RFID, GPS, sensors, actuators) and share this data with other Agents to offer real-time adaptability in (demand) supply chain (networks).

This concept is the core of Multi-Agent System. Real-time adaptability may affect a vast array of static or pre-set business processes. It is likely that many processes may change to evolve into the paradigm shift that is leading toward the adaptable business network (ABN). In particular, real-time adaptability may revolutionize supply chain management, fostering supply chain innovation through deployment of Multi-Agent Systems. Agent-based models draws clues from natural behaviour of biological communities of ants, wasps, termites, birds, fishes and wolves, to name a few.

In commercial supply chain software (i2, SAP, Oracle) processes are defined in terms of rates and flows (consumption, production). System variables (cost, rebates, transportation time, out-of-stock) evaluate or integrate sets of algebraic equations (ODE, ordinary differential equations or PDE, partial differential equations) relating these variables to optimise for best results (best price, shortest lead time, minimal inventory). The process (EBM, equation-based modeling) assumes that these parameters are linear in nature and relevant data are available. In the real world, events are non-linear, actions are discrete and information about data is distributed.

Solutions shall emerge where Agents-based software may function continuously and autonomously in an environment (Multi-Agent Systems) and processes. Continuity and autonomy indicates that Agents are able to execute processes or carry out activities in a flexible, intelligent manner that is both adaptive and responsive to changes in the environment without requiring constant human guidance, intervention or top-down control from a system operator. An Agent that functions continuously in an environment over a period of time may “learn” from experience (patterns). Agents that inhabit an environment with other Agents (Multi-Agent Systems) are able to communicate, cooperate, are mobile (one environment to another). The mobile, networked, autonomous, self-learning, adaptive Agent may have radically different principles compared to those that were developed for monolithic systems. Examination of naturally occurring Agent-based systems suggests design principles for the next generation of Agents. While particular circumstances may warrant deliberate exceptions, in general:

[1] Agents should correspond to “things” in the problem domain rather than to abstract functions.
[2] Agents should be small in mass, time (able to forget) & scope (avoid global knowledge action).
[3] Multi-Agent Systems should be decentralized (no single point of control/failure).
[4] Agents should be neither homogeneous nor incompatible but diverse.
[5] Agent communities should include a dissipative mechanism (entropy leak).
[6] Agents should have ways of caching and sharing what they learn about their environment.

Computer-based modeling has largely used system dynamics based on ODE. However, a multitude of industrial and businesses, including supply chain management, are struggling to respond in real-time. Eventually this transition may emerge as real-time adaptable business network. This paradigm shift will make it imperative to model software based on agents and equations. The question is no longer whether to select one or the other approach but to establish a mix of both and develop criteria for selecting one or other approach, that can offer real-time solutions. The “balance” is itself subject to change. For experts in supply chain management, the situation is analogous to “push-pull” strategy where the push-pull boundary may continuously shift with changing demand.
Difference in representational focus between ABM vs EBM has consequences for modularisation. EBM's represent the system as a set of equations that relate observables to one another. The basic unit of the model, the equation, typically relates observables whose values are affected by the actions of multiple individuals, so the natural modularisation often crosses boundaries among individuals. ABM represents the internal behaviour of each individual. An Agent’s behaviour may depend on observables generated by other individuals, but does not directly access the representation of those individuals’ behaviours, so the natural modularization follows boundaries among individuals. This fundamental difference in model structure gives ABM a key advantage in commercial supply chain management, in two ways:

First, in an ABM, each firm has its own Agents. Agents internal behaviours are not required to be visible to the rest of the system. Firms can maintain proprietary information about their internal operations. Groups of firms can conduct joint modeling exercises (MarketPlace) while keeping their Agents on their own computers, maintaining whatever controls are needed. Construction of EBM require disclosure of relationships that a firm maintains on observables so that equations can be formulated and evaluated. Distributed execution of EBM is not impossible, but does not naturally respect boundaries among the individuals (why public e-MarketPlaces failed to take-off). Second, in many cases, simulation of a system is part of a larger project whose desired outcome is a control scheme that more or less automatically regulates behaviour of the entire system. Agents correspond 1-to-1 with individuals (firms or divisions of firms) in the system being modeled, and their behaviours are analogs of the real behaviours. These two characteristics make Agents a natural locus for the application of adaptive techniques that can modify their behaviours as Agents execute, so as to control emergent behavior of the overall system. Migration from simulation model to adaptive control model is much straightforward in ABM than EBM. One can imagine a member of adaptable business network or supply chain using its simulation Agent as the basis for an automated control Agent that handles routine interactions with trading partners. It is less likely that such a firm would submit aspects of its operation to an external “equation manager” that maintains specified relationships among observables from several firms.

ABM’s support more direct experimentation. Managers playing “what-if” games with the model can think directly in terms of familiar business processes, rather than having to translate them into equations relating observables. ABM’s are easier to translate back into practice. One purpose of “what-if” experiments is to identify improved business practices that can be implemented. If the model is expressed and modified directly in terms of behaviours, implementation of its recommendations is a matter of transcribing the modified behaviours of Agents into task descriptions for the underlying physical entities in the real world. Disadvantages of EBM result largely from use of averages of critical system variables over time and space. EBM assume homogeneity among individuals but individuals in real systems are heterogeneous. When the dynamics are non-linear, local variations from the averages can lead to significant deviations in overall system behavior. In business applications driven by ‘if-then’ decisions, non-linearity is the rule. Because ABM’s are inherently local, it is natural to let each Agent monitor the value of system variables locally, without averaging over time and space and thus without losing the local idiosyncrasies that can determine overall system behavior.

The approach to system design and supply chain management with Agents in the software landscape is at odds with the centralized top-down tradition in current systems. The question usually arises in terms of the contrast between local and global optimization. Decision-makers fear that by turning control of a system over to locally autonomous Agents without a central decision-making body, they will lose value that could have been captured by an integrated (enterprise) global approach. The benefits of Agent-based architecture approaches vs centralized ones are conditional, not absolute. In a stable environment, a centralized approach can be optimized to out-perform initial efforts of an opportunistic distributed system of Agents. If the distributed system has appropriate learning capabilities, it will eventually become as efficient. Market conditions are marked by rapid and unpredictable change, not stability. Change and contingency are inescapable features of the real world. The appropriate comparison is not between local and global optima but between static versus adaptable systems. Real-time adaptability is crucial to supply chain management.
Executive Definition of AGENTS

A software entity that functions *continuously and autonomously* in a particular environment, often inhabited by other Agents and processes. Continuity and autonomy indicates that Agents are able to execute processes or carry out activities in a flexible and intelligent manner that is both *adaptive* and responsive to changes in the environment without requiring constant human guidance, intervention or top-down control from a system operator. An Agent that functions continuously in an environment over a period of time would be able to *learn from experience* (patterns). In addition, Agents that inhabit an environment with other Agents *(Multi-Agent Systems)* are able to communicate, cooperate and are *mobile* (from one environment to another). This simple definition of mobile, networked, autonomous, self-learning, adaptive Agent captures the essence but barely hints at the depth of Agent technology.

Executive Summary of AGENTS

Trends in software architecture may lead to widespread use of software Agents. Some programmers are designing Agents with the same principles that were developed for monolithic systems. Examination of naturally occurring Agent-based systems (ant) suggests radically different design principles for the next generation of Agents. While particular circumstances may warrant deliberate exceptions, in general:

1. Agents should correspond to “*things*” in the problem domain rather than to abstract functions.
2. Agents should be *small in mass*, time (able to *forget*) & scope (avoid global knowledge action).
3. Agent community *(Multi-Agent System; Complex Adaptive System)* should be *decentralized* (no single point of control/failure).
4. Agents should be neither homogeneous nor incompatible but diverse. Randomness and repulsion are important tools for establishing and maintaining this *diversity*.
5. Agent communities should include a dissipative mechanism (to whose flow Agents can orient themselves) providing *entropy leak* away from the macro level at which they do useful work.
6. Agents should have ways of caching and sharing what they *learn* about their environment, whether at the level of the individual, generational chain the overall community organization.
7. Agents should *plan and execute concurrently* rather than sequentially.
Introduction

You spill a grain of sugar on the table while adding a spoonful to your cup of tea. A harmless, small black ant keeps you company while it loiters on the table. Your flight is delayed and you have time to kill. You start observing the ant and wonder why it meanders so much in its circuitous quest for that grain of sugar. Through evolution, you begin to marvel, we, humans, have developed such superior intelligence when compared to the behaviour of this ant, on your table, trying to reach the grain of sugar in such a round-about, energy inefficient way. Stupid ant, you think.

Think, again!

Although it still remains a paradox, it is increasingly undeniable that simple individual behaviours of bugs like ants and wasps, collectively, may offer intelligent models of complicated overall behaviour. In fact, this may have been known for centuries. One ancient observer, King Solomon, knew from his father, David, of the elaborate court organizations of oriental kings and preparations needed for military campaigns. He marveled that insects could accomplish both these tasks without any central control. Thinking of the complex systems needed to maintain the palace commissary, he wrote, “Go to the ant, consider her ways and be wise. Having no guide, overseer or ruler, she prepares her bread in summer and gathers her food at harvest time.” He knew the complexity of a military organization and was impressed that “locusts have no king, yet all of them go forth by companies.” Nearly 3000 years later, a participant in the NCMS Virtual Enterprise Workshop (1994) commented, “we used to think that bugs were the problem. Now we suspect they may be the solution!”

Timeline of Agents Research

The idea of Agent originated with John McCarthy in the 1950’s at MIT. The term “Agent” was coined by Oliver Selfridge, a few years later, at Massachusetts Institute of Technology (MIT), where McCarthy and Selfridge were colleagues. The recent trends in Agent research began in 1977 and championed by Nicholas Negroponte of MIT Media Lab since 1990. Research from US Department of Defense (DARPA), Carnegie-Mellon University and University of Michigan at Ann Arbor has made significant contributions.

Why are Agents Necessary?

In current software, we define most processes as rates and flows (consumption, production). We define system variables (cost, rebates, transportation time, out-of-stock) and evaluate or integrate sets of algebraic equations (ordinary differential equations or partial differential equations) relating these variables to optimize for best results (best price, shortest lead time, minimal inventory). The process assumes that these parameters are linear in nature and relevant data are available. In the real world, events are non-linear, actions are discrete, and information about data is distributed.

Thus, lack of real-time adaptability is observed in models such as the Bull Whip Effect, which depicts wide fluctuations in supply chain causing severe financial detriment to businesses. The discrete, dynamic and distributed nature of data and applications require that software not merely respond to requests for information but also intelligently anticipate, adapt and actively support users. These systems should assist in coordinating tasks among humans and help manage cooperation among distributed programs. In response to these requirements, efforts of researchers from several fields are coalescing around a common broad agenda: the development of intelligent software Agents.
Agents, each supporting a clearly discernible task or process, interact with each other in a specified environment, for example, inventory control. Agents can interact with other Agents directly or via a message bus or through the knowledge base of the enterprise. Information Agents, for example, may continuously harness real-time data transmitted from sensors, actuators, radio frequency identification (802.11b, BlueTooth, 802.11a to ultra wide band RFID) tags, GPS and share this data with other Agents to offer real-time adaptability. Such real-time adaptability will profoundly affect a vast array of static business process. It is quite likely that many processes may require to be changed to evolve into the paradigm shift that is inevitably leading toward the adaptable business network (ABN). In particular, real-time adaptability may revolutionize supply chain management, fostering supply chain innovation through deployment of Multi-Agent Systems that draws clues, in part, from behaviours of biological systems or communities of ants, wasps, termites, birds, fishes and wolves.

How are Agent-based Models (ABM) different from Equation-based Models (EBM) ?

Virtually all computer-based modeling, up to this point has used system dynamics, an approach based on ordinary differential equations (ODE). Increasingly, a multitude of industrial and business processes, including amorphous systems like supply chain management, are struggling to be respond in real-time. Eventually and collectively this transition will emerge as the real-time adaptable business network (ABN). This paradigm shift will make it imperative to model business software based both with agents and equations. The question is no longer whether to select one or the other approach but to establish a business-wise mix of both and develop criteria for selecting balanced composition of software based one or other approach, that can offer valuable solutions. The “balance” is itself subject to dynamic changes. For experts in traditional supply chain management, the situation is analogous to a “push-pull” strategy where the push-pull boundary may shift with changing demand.

Both approaches simulate the system by constructing a model and executing it on a computer. The differences are in the form of the model and how it is executed. In Agent-based modeling (ABM), the model consists of a set of Agents that encapsulate the behaviours of the various individuals that make up the system and execution consists of emulating these behaviors, which is essentially dynamic. In equation-based modeling (EBM), the model is a set of equations (pre-determined static) and execution consists of evaluating them. Thus “simulation” is a general (umbrella) term that applies to both methods, which are distinguished as Agent-based emulation and equation-based evaluation.

Equation-Based Models (EBM)

Following the pioneering work of Jay Forrester (Unrelated to Forrester Research, Cambridge, MA) and the system dynamics movement, virtually all simulation work to date, principally on supply chains, integrates a set of ordinary differential equations over time. These models are graphically represented, using a notation that suggests a series of tanks connected by pipes with valves. The illustration below offers an example of material flow in a supply chain.

The rectangular boxes (Finished2, WIP2, Finished3, WIP3) are “levels” or variables whose assignments change over time. In this particular model, they represent four critical inventory levels (sites 2 and 3, in this model), a work-in-process (WIP) inventory and a finished goods inventory for each of the sites.
The **arrows with valve symbols** (shaped like hour-glasses: ship21; prod2; ship32; prod3; ship43) are flows between the levels they connect. The valves are "rates" or variables that determine the rate of flow. For example, the rate "prod2" is the rate at which site 2 converts work-in-process to finished goods.

The **cloud shapes** at the upper-left and lower-right corners represent external sources and sinks. In this case, upper-left corner represents end consumer (site 1), lower-right represents the supplier (site 4).

The **legends** associated with neither a box nor a valve (order rate, order period, production time, capacity) are auxiliary variables.

**Single-bodied arrows** show the dependency of rates on other variables (both levels and auxiliaries) in the model. The exact mathematical form of dependency is not shown in the graphical representation, but is encoded in the rate. For example, the arrows show that the rate "prod2" depends on the level "WIP2" and the auxiliary's production time and capacity.

This illustration models the components involved in the interplay between site capacity and order rate. When the order rate exceeds the site capacity, it demonstrates oscillations that, by extrapolation, may be representative of the fluctuations seen in the Bull Whip Effect. This system dynamics model shows the same periodicities as the Agent-based model but it does not show many of the effects that we observe in ABM and in real supply networks (memory effect of back orders, transition effects, amplification of order variation). Such effects require explicit representation of levels and flows for orders as well as parts. In particular, they require a model of PPIC (Production Planning and Inventory Control) in the system dynamics formalism (not easy to produce). System dynamics models of this nature are widely used in studying organizational dynamics, business processes, environmental and ecological systems, policy implications and a wide range of similar domains. In principle, ABM can be applied to all of these domains, often in a way that seems more natural.
Agents vs. Equations: A High-Level View

ABM and EBM share some common concerns but differ in two ways: the fundamental relationships among entities that they model and the level at which they focus their attention. Both approaches recognize that the world includes 2 kinds of entities: individuals and observables, each with a temporal aspect.

Individuals are bounded active regions of the domain. In some domains, the boundaries that set individuals apart are physical, as when we are studying ants or bees or people. In other domains, the boundaries may be more abstract, as in the case of the supply chain model, each representing a business firm. In any case, the boundaries are such that those who are interested in the domain recognize the individuals as distinct from one another. They are “active regions” because those interested in the domain conceive of the individuals as having behaviours. Individuals “do things” as time passes.

Observables are measurable characteristics of interest. They may be associated either with separate individuals (velocity of gas particles in a box) or with the collection of individuals as a whole (pressure in the box). In general, the values of these observables change over time. In both kinds of models, these observables are represented as variables that take on assignments.

Each of these sets of entities invites us to articulate the relationships that unify it and show how those relationships predict the behavior of the overall system through time. The first fundamental difference between ABM and EBM is in the relationships on which one focuses attention.

EBM begins with a set of equations that express relationships among observables. The evaluation of these equations produces the evolution of the observables over time. These equations may be algebraic or they may capture variability over time (ordinary differential equations, as used in system dynamics) or over time and space (partial differential equations). The modeler may recognize that these relationships result from the interlocking behaviours of the individuals but those behaviours have no explicit representation in EBM.

ABM begins, not with equations that relate observables to one another, but with behaviours through which individuals interact with one another. These behaviours may involve multiple individuals directly (foxes eating rabbits) or indirectly through a shared environment (horses and cows competing for grass). The modeler pays close attention to the observables as the model runs and may value a parsimonious account of the relations among those observables. However, such an account is the result of the modeling and simulation activity, not its starting point. The modeler begins by representing the behaviours of each individual, then turns them loose to interact. Direct relationships among the observables are an output of the process, not its input. The illustration below summarizes the critical relationships:

- Individuals are characterized, separately or in aggregate, by observables and affect the values of these observables by their actions.
- Observables are related to one another by equations.
- Individuals interact with one another through their behaviors.
A second fundamental difference between ABM and EBM is the level at which the model focuses. A system is made up of a set of interacting individuals. Some of the observables of interest may be defined only at the system level (pressure of an enclosed gas), while others may be expressed either at the individual level or as an aggregate at the system level (location of an organism vs the density of organisms per unit space of habitat). EBM tends to make extensive use of system level observables, since it is often easier to formulate parsimonious closed-form equations using such quantities. In contrast, the natural tendency in ABM is to define agent behaviours in terms of observables accessible to the individual Agent, which leads away from reliance on system-level information. In other words, the evolution of system-level observables does emerge from ABM but the modeler is not as likely to use these observables explicitly to drive the model’s dynamics as in EBM.

These two distinctions are tendencies, not hard and fast rules. The two approaches can be combined within an individual Agent in an ABM. Behaviour decisions may be driven by the evaluation of equations over particular observables and one could implement an Agent with global view whose task is to access system-level observables to make them visible to local Agents, thus driving an ABM with system level information. Furthermore, while Agents can embody arbitrary computational processes, some equation-based systems (those based on PDE’s, but not the simple ODE’s used in system dynamics) are also computationally complete. The decision between the two approaches, or a mix, must be made case by case, on the basis of practical considerations.

**Agents vs. Equations: Practical Considerations**

The difference in representational focus between ABM and EBM has consequences for how models are modularized. EBM’s represent the system as a set of equations that relate observables to one another. The basic unit of the model, the equation, typically relates observables whose values are affected by the actions of multiple individuals, so the natural modularization often crosses boundaries among individuals. ABM represents the internal behaviour of each individual. An Agent’s behaviour may depend on observables generated by other individuals, but does not directly access the representation of those individuals’ behaviours, so the natural modularization follows boundaries among individuals. This fundamental difference in model structure gives ABM a key advantage in commercial applications such as adaptable supply network modeling, in two ways:

First, in an ABM, each firm has its own Agents. An Agent’s internal behaviours are not required to be visible to the rest of the system, so firms can maintain proprietary information about their internal operations. Groups of firms can conduct joint modeling exercises (Public MarketPlace) while keeping their individual Agents on their own computers, maintaining whatever controls are needed.

Construction of an EBM requires disclosure of the relationships that each firm maintains on observables so that the equations can be formulated and evaluated. Distributed execution of EBM is not impossible, but does not naturally respect commercially important boundaries among the individuals (why MarketPlaces failed to take off).

Second, in many cases, simulation of a system is part of a larger project whose desired outcome is a control scheme that more or less automatically regulates the behaviour of the entire system. Agents correspond one-to-one with the individuals (firms or divisions of firms) in the system being modeled, and their behaviours are analogs of the real behaviours. These two characteristics make Agents a natural locus for the application of adaptive techniques that can modify their behaviours as the Agents execute, so as to control the emergent behavior of the overall system. Migration from simulation model to adaptive control model is much straightforward in ABM than in EBM. One can imagine a member of adaptable business network or supply chain using its simulation Agent as the basis for an automated control Agent that handles routine interactions with trading partners. It is much less likely that such a firm would submit aspects of its operation to an external “equation manager” that maintains specified relationships among observables from several firms.
Generally, ABM’s are better suited to domains where the natural unit of decomposition is the individual rather than the observable or the equation and where physical distribution of the computation across multiple processors is desirable. EBM’s may be better suited to domains where the natural unit of decomposition is the observable or equation rather than the individual.

EBM most naturally represents the process being analyzed as a set of flow rates and levels. ABM most naturally represents the process as a set of behaviours, which may include features difficult to represent as rates and levels, such as step-by-step processes and conditional decisions. ODE’s are well-suited to represent purely physical processes. However, business processes are dominated by non-linear, discrete decision-making.

ABM’s are easier to construct. Certain behaviours are difficult to translate into a consistent rate-and-level formalism. PPIC algorithms are an example. A recent release of a system dynamics package includes “black boxes” for specific entities such as conveyors or ovens whose behaviour is difficult to represent in a pure rate-and-level system. One suspects that the only realistic way to incorporate complex decision algorithms such as PPIC in system dynamics models will be by implementing such black boxes, thus incorporating elements of ABM, in the spirit.

ABM’s make it easier to distinguish physical space from interaction space. In many applications, physical space helps define which individuals can interact with one another. Customer-supplier relationships a century ago were dominated by physical space, leading to the development of regional industries, such as the automotive industry in southeast Michigan. Advances in telecommunications and transportation enable companies that are physically separate from one another to interact relatively easily, so that automotive suppliers in Michigan now find themselves in competition with suppliers based in Mexico or the Pacific rim. Such examples show that physical space is an increasingly poor surrogate for interaction space in applications such as commerce. In EBM, ODE methods such as system dynamics have no intrinsic model of space at all. PDE’s provide a parsimonious model of physical space, but not of interaction space. ABM’s permit the definition of arbitrary topologies for Agent interactions.

ABM’s offer an additional level of validation. Both ABM’s and EBM’s can be validated at the system level by comparing model output with real system behavior. In addition, ABM’s can be validated at the individual level, since the behaviours encoded for each Agent can be compared with local observations on the actual behaviour of the domain individuals.

ABM’s support more direct experimentation. Managers playing “what-if” games with the model can think directly in terms of familiar business processes, rather than having to translate them into equations relating observables.

ABM’s are easier to translate back into practice. One purpose of “what-if” experiments is to identify improved business practices that can be implemented. If the model is expressed and modified directly in terms of behaviours, implementation of its recommendations is a matter of transcribing the modified behaviours of Agents into task descriptions for the underlying physical entities in the real world.

In many domains, ABM gives more realistic results than EBM, for manageable levels of representational detail. The qualification about level of detail is important. Since PDE’s are computationally complete, in principle, one can construct a set of PDE’s that completely mimics the behavior of any ABM (thus produce the same results). However, the PDE model may be much too complex for reasonable manipulation and comprehension. EBM’s (like system dynamics) based on simpler formalisms than PDE’s may yield less realistic results regardless of the level of detail in the representation.

For example, the dynamics of traffic networks achieved more realistic results from traffic models that emulate the behaviours of individual drivers and vehicles, compared with the previous generation of models that simulate traffic as flow of a fluid through a network. The latter example bears strong similarities to the flow-and-stock approach to supply chain simulation.
The disadvantages of EBM in this and other examples result largely from the use of averages of critical system variables over time and space. EBM assumes homogeneity among individuals but individuals in real systems are often highly heterogeneous. When the dynamics are non-linear, local variations from the averages can lead to significant deviations in overall system behavior. In business applications, driven by “if-then” decisions, **non-linearity is the rule**. Because ABM’s are inherently local, it is natural to let each Agent monitor the value of system variables locally, without averaging over time and space and thus without losing the local idiosyncrasies that can determine overall system behavior.

**Evolution of Agent-Oriented Programming**

The history of software is one of increasing localization and encapsulation. Originally, the basic unit of software was the complete program. Arbitrary jumps of control made it difficult to manipulate any unit other than the individual line of code and the entire program. Data often occupied the same deck of cards as the program and the entire deck (code and program) was the responsibility of programmer, who thus determined the behavior of the complete (“monolithic”) program before it began execution.

The “structured” programming movement designed programs from smaller packages of code, such as structured loops and subroutines, with a high degree of local integrity. Though a subroutine’s code was encapsulated, its state had to be supplied externally through arguments and it gained control only when externally invoked by a call.

The next generation was “object-oriented” programming, which localized not only a segment of code but also the variables manipulated by that code. Originally, objects were passive and gained control only when some external entity sent them a message. Objects are visible, have specific locations and may contain things.

Agent’s architecture gives each object its own thread of control and its internal goals, thus localizing not only code and data, but also invocation. Such an “active object with initiative” is the most basic manifestation of Agent software, sometimes called an “Autonomous Agent” to emphasize that it does not need external invocation. It is also referred to as “responsible agent” to emphasize that it watches out for its own set of internal responsibilities. With code, state and control all localized within the Agent, little or no integration is required to launch an application. Agents can notice things, can carry out actions, can go places (environment) and can learn (therefore, adapt and respond to environment).

<table>
<thead>
<tr>
<th>How does a unit behave? (Code)</th>
<th>Monolithic Program</th>
<th>Structured Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent-Oriented Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>Local</td>
<td>Local</td>
<td>Local</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What does a unit do when it runs? (State)</th>
<th>Monolithic Program</th>
<th>Structured Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent-Oriented Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>External</td>
<td>Local</td>
<td>Local</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>When does a unit run?</th>
<th>Monolithic Program</th>
<th>Structured Programming</th>
<th>Object-Oriented Programming</th>
<th>Agent-Oriented Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>External (called)</td>
<td>External (called)</td>
<td>External (message)</td>
<td>Local (rules; goals)</td>
<td></td>
</tr>
</tbody>
</table>

In the ultimate Agent vision, the application developer simply identifies the specific mixture of Agents required or desired for each of the processes in the final application. The Agents organize themselves (Multi-Agent System) to perform the required functionality.
Computational Multi-Agent Systems: Concepts

One hundred agents, each with 10 behaviors, require programming of 1000 individual behaviors, yet provide a behaviour space of $10^{100}$ which is a number far larger than total number of elementary particles in the universe! Integration and maintenance costs, traditionally two of the largest expenses in software development, are greatly reduced.

Some evidence also suggests that interacting systems of Agents (Multi-Agent Systems), in addition to adaptability, can also offer robustness only hypothesized in traditional software systems. Appeal of Agent architectures depends on the ability of populations of Agents to organize themselves and adapt dynamically to changing circumstances without top-down control from a system operator.

A comparison between natural agent systems such as ant colonies and systems of computerized Agents depends on a model of “agent-hood” that is sufficiently general to cover both cases. Such a common model shows us important points of difference between the two domains, differences that we must take into account in engineering effective Multi-Agent Systems. A fundamental insight is that Agents cannot be considered independently of the environment in which they exist and through which they interact. In engineering Agents for non-electronic domains (factory automation) we must consider the environment explicitly.

Multi-Agent System (MAS) may be defined as a three-tuple (Agents, Environment and Coupling between them):

$$\text{MAS} = \langle \text{Agents, Environment, Coupling} \rangle$$

Each agent is a four-tuple:

$$\text{Agents} = \{\text{Agent}_1, \ldots, \text{Agent}_n\}$$

$$\text{Agent}_i = \langle \text{State}_i, \text{Input}_i, \text{Output}_i, \text{Process}_i \rangle$$

$\text{State}_i$ is a set of values that completely define the Agent. The structure, domains and variability of these values are not constrained by this definition. Differences in these features are responsible for much of the interesting variation among different kinds of Agents.

$\text{Input}_i$ are subsets of State, whose variables are coupled to the environment (an Agent’s sensors and effectors). These are mechanisms that implement the coupling between the environment and the Agent’s Input and Output variables, respectively.

$\text{Process}_i$ is an autonomously executing mapping that changes the Agent’s State (Process runs without being invoked from any outside entity). In computational terms, an Agent has its own virtual CPU.

The existence of Input and Output imply that an Agent is a bounded process. Input and Output cross this boundary. Input and Output relate the Agent immediately to the environment and only mediately (through the environment) to other Agents.

Environment is a two-tuple that is syntactically a subset of an Agent:

$$\text{Environment} = \langle \text{State}_e, \text{Process}_e \rangle$$
The important feature of this definition of the environment is that the environment itself is active. It has its own Process that can change its State, independent of the actions of embedded Agents. \( Input_i \) and \( Output_i \) of the various Agents are coupled to elements of \( State_e \), but the environment does not distinguish which elements of \( State_e \) are coupled. That distinction depends on the Agents that exist at any moment and the capabilities of their sensors and effectors. The lack of a distinguished Input and Output means that the environment, unlike an Agent, is unbounded. The fact that environment, like Agent, has State and Process means that putting a boundary and associated Input and Output values around an environment (and its associated Agents) yields a higher-level Agent. Thus this model lends itself naturally to aggregating low-level Agents into higher-level Agents creating the “World Model Agent” or similar concepts that is key to scalability.

**Agents are NOT Functions, Agents are THINGS**

Classical software engineering lead system designers toward “functional” decomposition. For example, manufacturing information systems contain modules for typical functions such as scheduling, material management, maintenance. In Agent-based systems, these instincts lead to individual Agents assigned to functions as Logistics, Transportation Routing, Resource Management, Scheduling, Dispatching. The functional approach is well suited to centralized systems but unprecedented in naturally occurring systems, which divide Agents on the basis of distinct entities in the physical world rather than functional abstractions in the mind of the designer. A functional decomposition appears most natural when the distinction between Agents and their environment is overlooked. A clear distinction between Agent and environment (such as that forced by discrete-event dynamics in the Agent and time-based dynamics in the environment) and a recognition that the environment’s own process mediates Agent interactions, forces the designer to pay attention to topological boundaries between Agents and environment and makes it more difficult to assign Agents to arbitrarily conceived functions.

In natural systems functions are important but they emerge from interactions of individual components rather than being represented as units in their own right. The temptation to functional decomposition is reflected in the fallacy of identifying a traffic jam with a collection of cars. Recognizing that fallacy is an important first step to avoiding the error. Aggregation of Agent behaviours rather than Agents themselves, also focuses on this issue. Experience with Agent-based prototypes suggests that each functional Agent needs detailed knowledge of many of the physical entities being managed. When the physical system changes, functional Agent needs to change as well. However, it is often possible to define generic behaviours for physically defined Agents from which required functionality will emerge.

There are two important exceptions to this principle: legacy systems and watchdogs. Most industrial Agent applications are additions to existing systems. They need to interface with legacy systems, many of which are functionally oriented. For example, a shop-floor control system needs to interface with a factory-wide MRP system performing classical scheduling. A reasonable way to connect legacy system to the new system is to encapsulate it as an Agent (see case study in aircraft maintenance). Even though MRP system may be functionally defined as a legacy program it is a well-defined “thing” in the application domain and so it deserves Agent-hood. Its presence in the system will not preclude the development of techniques for emergent scheduling as long as its Agent behaviours restrict it to serving as a link with the rest of the system rather than drawing on its centralized scheduling behaviour.

Some system states, that local agents cannot perceive, need to be monitored to ensure overall system safety or performance. A **Watchdog Agent** that simply monitors behaviour of a population of physical Agents is not nearly as restrictive on future re-configuration as one that does centralized planning and action. It is best not to rely on watchdogs at all, but if they are used, they should sense conditions and raise signals but not plan or take action.
Ovals and rectangles (figure above) show the Agent classes in an Agent community. Each material handling resource (truck or fork-lift), each processing resource (machine tool or operator) and each part is a separate Agent. There are no separate Agents for such standard functions as scheduling or logistics, but these functions emerge from the interaction of the local Agents. The Manager Agent is an example of a Watchdog Agent, who maintains a global view of the operation and raises signals to alert the other Agents when something undetectable on the local level needs attention.

Recently, intelligent PLM monitoring systems in factory automation have implemented the Watchdog Agent concept that sends signals or alerts based on feedback from sensors and monitoring devices.
Fundamentals of Agents: Model that draws from Natural Behaviour of Insects

Agents-based software architecture may be viewed as a reductionist approach to deal with equation-based software complexity that continues to increase exponentially. On first blush, it may seem like a fallacy to model simple behaviours of ants and other insects as a solution for reducing complexity. Although it still remains a paradox, it is undeniable that simple individual behaviours of ants, wasps, collectively, may offer intelligent models of complicated overall behaviour.

Part of the answer lies in the fact that teamwork of social insects is decentralized. An ant’s individual solution is primitive but collectively it results in efficient solutions to complex problems. The “active” key to ant’s efficiency is deposition of pheromone along their trails in their communal travel.

Pheromones are a medley of cyclical biological macromolecules affecting olfactory and other neuro-sensory pathways to elicit diverse responses in insects, humans, yeasts. Some pheromones may belong to arachidonic acid derivatives generically referred to as Prostaglandins.

For example, advances in data routing emerged from study of ant-based algorithms, based on concept of swarm intelligence. To build better data networks, “artificial ants” created as software, travel through network/programs depositing “artificial pheromones” pursuing route optimisation. Bending the rules of this ant behaviour is a primary step. Bending the rules involve endowing the “soft-ants” with memory (parameters, variables, formulas) and enabling them to retrace “good routes” and mark them with extra “artificial pheromones” that then may be extrapolated to routing, marketing and event management strategies.

Ant-based algorithms are also being borrowed to solve a classic puzzle known as the traveling salesman problem. It can be also related to systems, goods, raw materials, parcels, packets of data. As the number of cities involved increases in the traveling salesman problem, the difficulty of the problem increases exponentially. In one model, artificial pheromone deposited along longer routes evaporate, leaving the links to the greatest number of shortest routes most densely covered with the artificial pheromone. When soft-ants are sent out again, they rely on “memory” tables storing information about amount of pheromone on each link. Repeating these trips result in progressively shorter overall trips.

The latter may be particularly relevant to unpredictability in data flow, such as “Bull Whip Effect” for supply chain volatility or internet data traffic. Because soft-ants are constantly exploring different routes, many alternatives surface if a particular “good” route goes out of commission. The processing ability in this model is based on numbers. If there are 5 ants in a “colony” then this complex processing fails but if you have 10,000 “soft-ants” then the results are magical.

In complex distributed systems, such as, supply chain network and event management, where routing and sequence is critical, current solutions are offered from people aiming to optimize interactions with a centralized mind-set. Swarm intelligence based decentralized control underlies the paradigm shift for optimisation in future adaptable business networks. These models, as key principles in Agents technology, are able to find very good solutions (but may not be the perfect optimisation) in a reasonably short time. Not provably optimal but that which is “very good” is often required for real-world applications in real-time.

Ants construct networks of paths that connect their nests with available food sources. These networks form minimum spanning trees, minimizing the energy ants expend in bringing food into the nest. Graph theory defines a number of algorithms for computing minimum spanning trees, but ants do not use conventional algorithms! Instead, this globally optimal structure emerges from the simple actions of individual ants. Each ant that forages for food has the same basic program, consisting of five rules that fire whenever their conditions are satisfied:
[1] Avoid obstacles. Whatever an ant does, it will not aimlessly push against a wall.

[2] Wander randomly in the general direction of nearby pheromones. If it does not sense pheromones, an ant executes Brownian motion, choosing each step from a uniform distribution over possible directions. If it senses pheromones, the ant continues to wander randomly but the distribution from which it selects its direction is weighted to favor the direction of the pheromone scent.

[3] If the ant is holding food, it drops pheromone at a constant rate as it walks. In simplest simulations, the ant continues to move randomly. In others, it follows a beacon (distinctive pheromone at the nest) that leads in the general direction of home. Both approaches yield same global behavior. The homing beacon generates paths sooner but continued random wandering works in the emulation as well.

[4] If the ant finds itself at food and is not holding any, it picks up the food.

[5] If the ant finds itself at the nest and is carrying food, it drops the food.

Brownian motion eventually brings the ant arbitrarily close to every point in the plane. As long as the separation between nest and food is small enough compared with the range of the ant, a wandering ant will eventually find food if there is any and (even without a beacon) a food-carrying ant will eventually find the nest again. In most cases, food is available only in some directions from the nest and ants who wander off in the wrong direction will starve or fall to predators. But, as long as there is food close enough to the nest and as long as there are enough ants to survey the terrain, the food will be found.

Only food-carrying ants drop pheromone and because ants can carry food only after picking it up at a food source, all pheromone paths lead to a food source. Once a full ant finds its way home, there will be paths that lead home as well. Because pheromones evaporate, paths to depleted food sources disappear, as do paths laid down by food-carrying ants that never reach home. Paths that touch the nest are easily found by out-bound ants. As long as they lead to food, they will be reinforced by those ants once they pick up food.

The initial path will not be straight but the tendency of ants to wander even in the presence of pheromones will generate short-cuts across initial meanders. Because pheromone paths have some breadth, they tend to merge together into a trace that becomes straighter, the more it is used. The character of resulting network as a minimal spanning tree is not intuitively obvious from individual behaviours, but does emerge from the emulation.

An ant hill houses different kinds of things, including larvae, eggs, cocoons and food. The ant colony keeps these entities sorted by kind. For example, when an egg hatches, the larva does not stay with other eggs but is moved to the area for larvae. Computer science has developed many algorithms for sorting things, but ants in the ant hill are not executing a sorting algorithm!

(Could it explain why retail cross-docking experts like P&G & W*M may be developing ant affinity ?)

Individual ant algorithms that yields system-level sorting behaviour contains some behaviours similar to those in the path-planning problem:

1. Wander randomly around the nest.
2. Sense nearby objects and maintain a short memory (about ten steps) of what has been seen.
3. If an ant is not carrying anything when it encounters an object, decide stochastically whether or not to pick up the object. The probability of picking up an object decreases if the ant has recently encountered similar objects.
4. If an ant is carrying something, at each time step decide stochastically whether or not to drop it, where the probability of dropping a carried object increases if the ant has recently encountered similar items in the environment.
As in path planning, Brownian walk eventually brings the wandering ants to examine all objects in the nest. Even a random scattering of different items in the nest will yield local concentrations of similar items that stimulate ants to drop other similar items. As concentrations grow, they tend to retain current members and attract new ones. The stochastic nature of the pick-up and drop behaviours enables multiple concentrations to merge, since ants occasionally pick up items from one existing concentration and transport them to another. The put-down constant $k^-$ must be stronger than the pick-up constant $k^+$ or else clusters will dissolve faster than they form. Typically, $k^+$ is about 1 and $k^-$ is about 3. The length of short-term memory and the length of the ant’s step in each time period determine the radius within which the ant compares objects. If the memory is too long, the ant sees the entire nest as a single location and sorting will not take place. The limited short-term memory of ants ensures that ants forget.

The “forgetful” nature of ants may seem too esoteric for real-world applications. In reality, the ant’s ability to “forget” is a boon to real-world adaptable business networks if impregnated with software Agents. In traditional equation-based planning algorithms, demand forecasting is based on a weighted-average of past consumption data. If there was an anomaly, for example, spike of sales 20 weeks ago that was a variant, the planning algorithm continues to consider that value because equation-based modeling cannot “forget” facts, although the weight will decrease with length of time. The forecasting engine, therefore, continues to reflect the effect in its subsequent forecast for many weeks or years unless the parameters are manually changed. Such events alone, sprouting up in so many places within the supply chain, produce forecasts that are wrong, adding to the cascade of other events that causes wide and apparently uncontrollable fluctuations observed in the supply chain (Bull Whip Effect).

Ant-based algorithms, utilizing the concept of swarm intelligence, in Agents technology, enables Agents to “forget” since Agent behaviour is modeled on natural systems behaviour rather than equation-based models of current software. The ability to learn and to forget, enables Agents to be adaptive. Such adaptive Agents, in the example above, will enable planning and forecasting to be much more accurate. The accuracy may be even more enhanced by real-time data acquired by an information Agent (say, from RFID tags). Real-time data-catalysed adaptability of Agents may (vastly) improve forecasting and reduce inventory. Reduced inventory reduces working capital charges and contributes to reduced production waste. These reductions improve return on assets because manufacturing cash cycle gets shorter.

**Task Differentiation in Wasp Communities may offer clues to HR & Labour Resource Planning**

Mature Polistes wasps in a nest divide into three groups: a single Chief, a group of Foragers who hunt for food and a group of Nurses who care for the brood. These varied roles are filled by genetically identical wasps. The relative proportion of Foragers and Nurses varies with the abundance of food and the size of the brood. The nest has no Human Resources and no wasp (not even the Chief) computes what this proportion should be.

Each wasp maintains a [1] *Force* parameter that determines its mobility and [2] a *Foraging Threshold* that determines how likely the wasp is to go seek for food. The brood maintains a third parameter, *Demand*, which stimulates the Foragers. Wasp behaviours involve interactions of these parameters:

[1] When two wasps meet, they engage in a face-off. The winner is chosen stochastically. Wasp with higher force has a higher probability of winning but the wasp with lower force will occasionally win. A quantum of force is transferred from the loser to the winner wasp. (Mathematically, the probability of success of individual 1 in confronting individual 2 is given by the Fermi function of their forces where a constant ($k$) modulates the extent to which the outcome is predetermined.)


[3] A wasp near the brood, determine probabilistically whether or not to forage. Successful stimulation reduces foraging threshold by a certain constant, the learning coefficient, while failure to forage increases foraging threshold by a constant, the forgetting coefficient.
Confrontation among wasps shifts force from one to another and represents a form of communication. Foraging reduces brood’s demand and thus brood’s stimulation on nearby wasps. Stimulation reduces wasps’ thresholds and triggers foraging. When a community of “soft-wasps” executes these behaviors over a period of time, the population stabilizes into 3 groups, corresponding to the division observed in the natural insects. A group with high force and low threshold corresponds to the Foragers, who have both the strength to move about and the sensitivity to the brood to respond to their needs. A second group with low force and low threshold corresponds to the Nurses, who also are attentive to the brood but, lacking force, cannot move about and must remain near the brood. Finally, a single wasp with high Force and high threshold corresponds to the Chief. The Chief does not command or control the others but grounds the force and threshold scales (by wandering around the nest and facing off against the other wasps) and balances these variables across the population.

Local and Global Optimisation through Agent Behaviour

Flocks of birds stay together, coordinate turns, avoid collisions with obstacles and each other. Schools of fish exhibit similar coordinated behaviour. Humans address similar problems, in air-traffic control and convoys of ships but conventional solutions depend on sophisticated communication and central coordination structures. Most sophisticated human coordination cannot handle density of coordinated entities of fishes or birds. Each bird or fish follows only 3 simple rules to achieve such coordination:

[1] Maintain a specified minimum separation from the nearest object (other birds, fish)
[2] Match velocity (magnitude and direction) to nearby objects
[3] Stay close to the center

The flock or school is a self-constraining structure in which each entity's individual actions (local) simultaneously respond to and change the overall structure of the flock (global). Although each bird or fish senses only the movements of its nearest peers, its responses to these movements propagate to others, so that the system as a whole exhibits global coordination.

Natural systems behaviour shares common principles of self-organization. As we learn to recognize and understand these principles, we construct artificial systems that emulate the desirable behaviour we observe in these natural systems. These are key to Agent engineering principles.

Properties shared by Multi-Agent Systems and “Guiding Heuristics”

Research programs devoted to understanding complex adaptive systems (CAS), analogous to Multi-Agent System (MAS), offers 4 properties that such systems share and 3 mechanisms that they employ. These features are analytic (not synthetic) and represent properties or mechanisms common to existing CAS or MAS. The four properties are:

[1] Aggregation
MAS are not constructed monolithically but consist of smaller components (“Agents”) which may be themselves aggregates of still smaller units (Agents). The behavior of the aggregate is often distinct from the individual behaviours of the parts.

[2] Non-linearity
The behavior of MAS is not linear and their interactions are, thus, not simply additive.

MAS are characterized by flows of various substances through networks of Agents. These flows exhibit two important effects: multiplication (in which one change produces a chain of others) and recycling (feedback loops).

[4] Diversity
The Agents in a MAS could differ from one another.

The three mechanisms are:

1. **Tagging**  
   Agents in a MAS are able to recognize and differentiate among one another (share information).

2. **Internal Models**  
   The internal structure of an Agent in a MAS enables it to anticipate changes in its environment. (share information).

3. **Building Blocks**  
   An Agent’s internal model is made up of small, reusable modules, thus enabling it to capture a rich set of alternatives with a limited representational vocabulary (share information).

The paradigm shift in software medium fostered by Agent technology requires new ways for people to think about decentralized systems. The following are “guiding heuristics” that people may use to learn to understand these systems:

1. Positive feedback is not always negative. Sometimes it leads to destructive oscillations but in other cases it is critical to self-organization (entropy).

2. Randomness can help create order by providing diversity among agents.

3. A flock is not a big bird. The behaviour of an aggregate system is not the same as the individual behaviours of the lower-level units out of which it is constructed.

4. A traffic jam is not just a collection of cars. Decentralized systems generate emergent objects that are distinct from any of the individual parts.

5. The hills are alive. The environment is an active process that impacts the behavior of the system, not just a passive communication channel between Agents.

6. Distribute. Emergent behavior is distributed across the components rather than localized in any single one. [Decentralize]

7. Persistent disequilibria. A useful system must balance stability with constant change, otherwise it will degenerate. [Entropy]

8. Change changes itself. In dealing with changing circumstances, complex systems change and over time the rules of change undergo change. [Share Information]

9. Think small. Keep Agents small, in mass, time and space (scope).

**Properties of Agents: Keep Agents Small (in Mass)**

Naturally occurring adaptive systems have parts that are small compared with the entire system, in mass, time and space. Tropical termites construct mounds that can exceed five meters in height and ten tons in mass. These multi-story structures store food, house the brood and protect the population from predators. The existence of some of these structures has been documented for over 350 years, which is as long as they have been accessible to the European compulsion for chronological records. In spite of the complexity, durability and effectiveness of these structures, no termite serves the role of a chief engineer to plan the structure and manage its construction. Termites draw on the pheromone mechanism as follows:
[1] Metabolize bodily waste, which contains pheromones. Excretory waste is the material from which termite mounds are constructed.

[2] Wander randomly but prefer the direction of the strongest local pheromone concentration.

[3] At each time step, decide stochastically whether to deposit current load of waste. The probability of making a deposit increases with the local pheromone density and the amount of waste that the termite is currently carrying. A full termite will drop its waste even if there is no other nearby deposit and a termite that senses a very high local concentration of pheromones will deposit whatever waste it is carrying, even a relatively small amount.

The probabilistic algorithm leads to the generation of scattered initial deposits. These deposits attract termites that wander close enough to smell them and increase the probability that these visitors will make reinforcing deposits. Because pheromones decay over time, the most recent deposits at the center of the pile are the strongest and thus the piles tend to grow upward rather than outward, forming columns. When two columns grow near one another, the scent of each attracts termites visiting the other, thus pulling subsequent deposits into the shape of an arch. A similar dynamic leads to the formation of floors joining multiple arches. When one floor is complete, the cycle repeats to construct the next. Each termite is an almost negligible part of the entire termite hill. As a result, the behavior of the whole is stable under variations in the performance of any single member. Collective dynamics dominate. This and similar examples suggest implementing artificial systems with large numbers of Agents, each small in comparison with the whole system.

The motivation for this principle derives not from the theory of multi-Agent systems, but from the experience of software engineers. Difficulty of designing, implementing and launching computer-based systems increases exponentially with the size of the system. Individual Agents are easier to construct and understand than large monolithic systems. Thus, the impact of the failure of any single Agent will be minimal. In addition, a large population of Agents gives the system a richer overall space of possible behaviours, thus providing for a wider scope of emergent behaviour. Roughly, the number of agents is a multiplicative factor in determining implementation effort, but an exponent in determining the size of the overall system state space (100 agents with 10 behaviors each = resulting state space of $10^{100}$).

Keeping agents small in mass often means favoring specialized Agents over more general ones, using appropriate aggregation techniques. For example, a complete manufacturing cell would be extremely complex as a single Agent, but can be developed as a community of Agents for individual mechanisms (one for the fixture, one for the tool, one for the load-unload mechanism, one for the gauging station). Multi-Agent system (MAS) directly supports such aggregation, since an environment (and its associated Agents) can become a higher-level Agent by defining its inputs and outputs to another environment.

Assignment of a separate Agent to each machine, each part, each tool, in a manufacturing enterprise provides Agents that are light weight than traditional shop-floor software systems. The automation (see Watchdog Agent) and industrial controls market offers increasing support for Agents as small as the sensor/actuator level, including small controllers of traditional design (Rockwell, Allen-Bradley, Mitsubishi) as well as more novel architectures combining computation, control and networking.

Properties of Agents: Keep Agents Small (in Time)

Naturally occurring Agent systems can forget (see example of supply chain elsewhere). Pheromones evaporate and obsolete paths leading to depleted food sources disappear rather than misleading members of the colony. The mechanism of forgetting is an important supplement to the emphasis in conventional artificial intelligence (AI) systems on mechanisms for learning. In a discrete-event system, forgetting can be as complex as learning since both represent discrete state transitions. In a time-based system, forgetting can take place “automatically” through the attenuation of a state variable that is not explicitly reinforced.
Pheromones suggest that a time-based environment can support a “forgetting” function for discrete-event agents. In one model, each segment of a switched conveyor system is a separate Agent that seeks to maintain its population of a given part type within certain limits. If the population rises above the upper limit, the segment seeks to spill excess parts to some neighbouring segment. If the population falls below the minimum, the segment sends requests out to neighbouring segments. The probability that a segment will spill a part to a given neighbour is a state variable for the spilling segment. It is increased each time the neighbour requests a part and decreased each time the segment spills a part to the neighbour. Because the information on a neighbour’s interest in a part is maintained as a real number rather than symbolically, obsolete behaviours are forgotten automatically as the value is modified to reflect more recent behaviour. The rate at which the probability changes in response to interactions with neighbouring segments is a tuning parameter.

**Properties of Agents: Keep Agents Small (in Scope, in Sensing Actions)**

Participants in natural systems usually sense their immediate (local) vicinity (birds, fishes). In spite of this restriction, they can generate effects that extend far beyond (global) their own limits, such as networks of ant paths or termite mounds. The exposition of the principle, “control from the bottom up” recognizes the superiority of many local interactions over a few global ones.

Bandwidth (WAN, LAN) makes it easy to connect each Agent directly with every other Agent but it may be beneficial to follow natural examples and engineer Agents that limit recipients of their messages. Telecommunications technology means that these limitations need not be geographical but natural examples suggest that effective systems will restrict communications in some way. Although exceptions will follow, where feasible and appropriate, Agents should define the audience that needs to receive the message, at least by subject-based addressing rather than broadcasting information.

A suggestive analogy comes from the history of physics. Newton’s classic equation for gravitational force usefully describes the effects of gravity but does not explain the underlying mechanism. Massive bodies do not figure out how hard to pull on one another by communicating their masses and measuring the distance that separates them. Einstein’s theory of general relativity replaces this impossible notion of action-at-a-distance with a model of local interaction in which masses warp space in their immediate vicinity and respond to the local geometry of the space in which they find themselves. Could localization of Agent interactions follow in the intellectual tradition of physics?

How ants sort their nests yields an interesting and related lesson. An ant whose short-term memory is too long “sees” the entire nest at once and is unable to sort. In simple terms, giving an Agent access to too much of the world may lead to sensory overload, reduced ability to discriminate and lower performance. In terms of the <Agents, Environment, Coupling> model, if Agents are small compared to the environment, their State will have fewer elements than the environment’s State. Their Input and Output subsets of State will be even smaller. The accuracy of the model of the environment presented by an Agent’s Input will begin to decrease once the Agent’s scope of sensing is so large that the portion of the environment accessible to it contains more information than its Input can model.

In this model, only environment has a state space large and rich enough to represent the entire system. This observation does not mean that we should neglect engineering the environment. The <Agents, Environment, Coupling> model emphasizes that we must pay attention to both components of the system. However, the rich information interface that computers support is available only through the Agents and there are important reasons not to make one Agent large and complicated enough to control the entire system:
[1] A central Agent is a single point of failure that makes the system vulnerable to accident.

[2] Under normal operating conditions, it can easily become a performance bottleneck.

[3] Even if adequately scaled for current operation, it provides a boundary beyond which the system cannot be expanded.

[4] It tends to attract functionality and code as the system develops, pulling the design away from the benefits of Agents and in time becoming a large software artifact.

Centralization can sometimes creep in when designers confuse a class of Agents with individual Agents. For example, one might be tempted to represent a bank of paint booths as “the paint Agent” because they all do the same thing. Certainly, one would develop a single class (in an object-oriented sense of the word) for paint-booth Agents but each paint booth should be a separate instantiation of that class.

The difference in size between an individual Agent’s State and that of the environment not only favours small Agents and decentralized control but also encourages Agent diversity. The environment’s State contains information concerning both opportunities that Agents should exploit and threats that they should avoid. The more of the environment’s State the Agents can sense and modify, the better they can exploit those opportunities and avoid threats. Any single Agent can model and manipulate only a small portion of the environment and a population of completely identical Agents will do no better since they will still cover only the same subset of the environment’s state. A population of diverse Agents will cover more of the environment’s state and thus provide better performance.

While diversity is not the same as population size, the two are correlated. In a physical environment, it is impossible for two Agents to occupy the same place at the same time. Thus two otherwise identical Agents will be at different locations in the environment. They will differ in that element of their respective States that records their location. This simple but crucial element of diversity enables them to monitor different elements of the environment’s state and collectively be more robust than a single Agent. The important observation is that the advantage of the larger population lies not merely in numbers but in the diversity that results from physical exclusion laws.

Natural populations often have a “critical size” that is much larger than the simple breeding pair that a standard accounting paradigm might justify. If the population falls below this level, the colony dies out. Once an ant finds food, it generates a pheromone trail to guide other ants to it, but the critical initial discovery depends on having enough ants wandering around that some will stumble across food, wherever it may be. Unused alternatives (unsuccessful scouts) are insurance for unforeseen change, not waste.

The example of similar Agents at different locations illustrates that diversity is not the same as incompatibility. The diversity in location among ants is able to benefit the society as a whole because in many other respects the ants are interchangeable. They all like the same kind of food and they all sense the same pheromones. Their diversity of location would be of no value if they were not similar enough to one another to share in the benefits that diversity conveys. Diversity can be mathematically quantified by formalizing these insights (beyond the scope of my abilities).

Naturally occurring random behavior can be a good model to base ways to achieve the diversity a population of Agents needs in order to adapt. Randomized agents attack a problem in a Monte Carlo fashion that does not require a detailed advance model of the domain and lend themselves to much simpler coordination than approaches that require an explicit reason for every decision. Compared with conventional systems, natural Agent-based systems may seem wasteful. They allocate far more resources to a task than a global analysis may require. For example, an ant’s ignorance of where food may be found requires a large army of scouts, some of whom wander far from both nest and food and either starves or are taken by predators. This example of ant alludes to redundancy.
Redundancy supports diversity in two ways. First, the diversity in location among the ants enhances the colony’s chances for finding food and thus surviving. Second, the small size of one ant in comparison with the colony means that several individuals can perish without endangering the entire community. These two insights reflect distinction among homogeneity, diversity and incompatibility. Diverse Agents are not homogeneous, they can monitor an environment that is much more complex than any single Agent. Because they are not incompatible, the community as a whole can tolerate the loss of any one individual, drawing on the overlapping capabilities of others.

Fish and bird examples show how repulsion among Agents can maintain diversity of location. The same technique could be applied to other aspects of the relation among Agents. In a machine job shop, a new job can be processed more rapidly if the necessary tooling is already on the machine. Machine Agents, representing otherwise identical machine tools in such a shop, might use a repulsive force with respect to their tooling characteristics to see to it that the currently unused machines have diverse tooling, thus increasing the chances that a new job will find a machine ready to run it.

Similarly, natural redundancy and its effect on diversity have useful real-world Agent applications. Sometimes lack of redundancy in the application domain leads to centralisation. If every product in a plant must pass through a single paint booth, it is all too easy to give the Agent representing that booth strong central characteristics. For example, with multiple paint booths, booths might bid for products, but with only one, it calls the shots unilaterally. In such a case, it is sometimes possible to reduce this effect by imagining that there are two or more such Agents and letting the “real” one regularly win run-time competitions. Only the “real” Agent would ever win a bid for work because the others would always report that their machines are off-line. If there is ever a need for new machines, they can be added without modifying the overall design.

Redundant agents can step in for one another because Agents may not communicate directly with one another, but only mediatly, by making changes in the environment, that are subsequently sensed by other Agents. Any Agent that senses these changes can respond. Failure of an individual Agent does not bring the system down as long as there are other Agents sufficiently similar to sense the same environmental changes and respond appropriately. Redundancy in production capability may be supported by using a negotiation protocol to identify potential suppliers for the inputs to a given unit process. For example, a unit process that needs a milling machine, issues a request indicating the required machine class in the subject, and through subject-based addressing, all machines of that class, that are currently on the network, will receive the request. Through a negotiation, the unit process discriminates among available machines on the basis of criteria such as operating cost and availability and selects a platform on which to execute. If one machine breaks down or is fully loaded, this approach permits another to take its place, as long as the physical shop has redundant capabilities.

The Second Law of Thermodynamics observes that closed systems progressively may become more disordered over time. It is not obvious that a large collection of Agents will organize itself to do useful things. The Second Law warns that the result of such an architecture may be disorder. Natural agent-based systems do organize themselves with striking efficiency. A common explanation is that a system can become more organized if energy is added to it from the outside (for example, by the metabolism of the food). The addition of energy is necessary for self-organization, but hardly sufficient. Gasoline in construction equipment can erect a building but the same gasoline in a terrorist’s bomb can destroy it.

In one model of this leakage, micro-level dissipation in the environment generates a flow field that the Agents can perceive and reinforce, and to which they can orient themselves. Insect colonies leak entropy by depositing pheromones whose molecules evaporate and spread through the environment.
under Brownian motion, generating entropy. The resulting flow produces a field that the insects can perceive and to which they orient themselves in making further pheromone deposits.

The above illustration (Macro Organization through Micro Dissipation) generalizes this example in terms of the three fundamental processes: micro dissipation, macro perception of the micro flow field and macro reinforcement of the micro dissipative mechanism.

One example is the movement of currency in a market economy. Money benefits its holders only when they spend it. As it spreads from purchasers to buyers, entrepreneurs perceive the resulting flow field and orient themselves to it, resulting in the self-organization of structures such as supply chains and geographic economic centers.

Artificial agent communities will be more robust and better able to organize themselves if they are designed to include a dissipative mechanism (entropy leak) such as a currency. This mechanism should have three characteristics:

1. It must flow, either among Agents or through the environment, thus, setting up a gradient field.
2. Agents must be able to perceive this field and orient themselves to it.
3. Agent’s actions must reinforce the field (positive feedback).

Properties of Agents: Learn and Share Information

Natural systems exchange information among members of the population at three levels: species, individual and society. Sexual reproduction exchanges information from one generation of a species to another by passing on successful characteristics in the form of chromosomes to offspring. Individual organisms can also pass on skills post-embryonically. For example, young black bears learn to rob food hung from trees by watching older bears. The society as a whole can learn even if individual members do not, as in the development of pheromone paths in insect colony. In each case, a community reduces the need for expensive search by finding ways to cache and share accumulated knowledge.
In the <Agents, Environment, Coupling> model, species and individuals learn by modifications to Agents’ State and Process components, while societies learn by modifying the environment’s State. Agents can often use similar mechanisms. While learning in a single Agent can require sophisticated techniques, methods for learning across generations, such as genetic or evolutionary programming, have proven their maturity in numerous applications. Changing a community’s structure to enable its members to respond better to a changed environment is straightforward for Agent architectures. In addition, Agents architecture can support genetic modification of various aspects of Agent behaviour.

Properties of Agents: Plan and Execute Concurrently

Traditional systems alternate planning and execution. A firm develops a schedule each night that optimizes its manufacturing the next day. Unfortunately, changes in the real world tend to invalidate advance plans. Engineers in industries as diverse as auto, semiconductors, aerospace and agricultural equipment agree that a daily schedule is obsolete less than an hour after the day begins.

The problem is a natural consequence of a system in which the environment as well as the Agents has a Process that can autonomously modify the environment State with which Agents interact. Natural systems do not plan in advance but adjust their operations on a time scale comparable to that in which their environment changes. The coherence of their behaviour over time is maintained by the dynamics of their interactions, not imposed by an external plan or schedule. To achieve the robustness exemplified by these natural systems, Agents should seek to avoid the “plan then execute” mode of operation and instead respond dynamically to changes in the environment.

Consider an example of concurrent planning and execution where the actual time at which a job will execute may not be known until the job starts. The resource does not schedule a newly-arrived job at a fixed point in time but estimates probabilistically the job’s impact on its utilization over time, based on information from the customer about the acceptable delivery times. The width of the window within which the job can be executed is incrementally reduced over time, as needed, to add other jobs to the resource’s list of tasks. If the resource is heavily loaded, the jobs organize themselves into a linear sequence but if it is lightly loaded, the actual order in which jobs are executed is decided at the moment the resource becomes available, depending on the circumstances that exist at that time.

Why Should We Consider “Out-of-the-Box” Thinking with Agents ?

The approach to system design and management with Agents in the software landscape is at odds with the centralized top-down tradition in systems engineering. The question usually arises in terms of the contrast between local and global optimization. Decision-makers fear that by turning control of a system over to locally autonomous Agents without a central decision-making body, they will lose value that could have been captured by an integrated (enterprise) global approach.

The benefits of Agent-based architecture approaches vs centralized ones are conditional, not absolute. In a stable environment, a centralized approach can be optimized to out-perform the initial efforts of an opportunistic distributed system of Agents. If the distributed system has appropriate learning capabilities, it will eventually become as efficient. Market conditions are marked by rapid and unpredictable change, not stability. Change and contingency are inescapable features of the real world. The appropriate comparison for systems designers or enterprise software is not between local and global optima but between static versus adaptable systems.

One should evaluate the competing options in a particular case theoretically, strategically, tactically, and practically.
Theoretically, there are decentralized mechanisms that can achieve global coordination. For example, economists have long studied how local decisions can yield globally reasonable effects. Recently these insights have been applied to a number of domains that were not traditionally considered as economic, such as network management, manufacturing scheduling and pollution control.

Strategically, managers must weigh the value of a system that is robust under continual change against one that can achieve a theoretical optimum in a steady-state that may never be realized. A company that anticipates a stable environment may well choose centralized optimisation. One that also incorporates Agent-based software does so because it cannot afford to be taken by surprise.

Tactically, the life-cycle software costs are lower for Agent-based systems than for centralized enterprise software because Agents can be modified and maintained individually at a fraction of the cost of opening up a complex enterprise software system. In systems that must be modified frequently, losses due to sub-optimal performance can recovered in reduced system maintenance expenses.

Practically, Agent-based systems that follow these principles have been piloted or deployed in regular operation. The Agents in these systems regularly reflect the principles outlined here rather than those of centralized systems. Growing acceptance of Agents technology in competitive business environments may be evidence of the benefit they bring to their adopters.

**What does the “Agent Vision” mean for ERP vendors (JBOPS)?**

Opportunity to provide thought leadership as well as concurrent market leadership through progressive and dynamic execution of Agents software to better facilitate real-time adaptable business networks.

Consider creating a purchase order via operation VA01. Creating PO (function) is a sequence of discrete processes that may begin by checking ATP, picking-packing, delivery, shipment. These processes are integrated, hard-coded and flexibility is pre-determined only by the parameters set in the IMG. To accomplish this function in the Agents “vision” the processes could be performed by an Agent. The collection of Agents to complete operation VA01 may be referred to as a Multi-Agent System for VA01. The “picking-packing” process, for example, may be accomplished by a Pick-Pack Agent. Since Agents are based on open architecture, it is possible that a tiny company of five developers in Putato, Burma, has produced the perfect “Pick-Pack Agent” which is deployed by Metro AG to accomplish its “pick-pack” process steps, where ever necessary, within its software infrastructure. LivingSystem, a tiny company from Medicine Hat, Canada, may have the best of breed “inventory” Agent used by Proctor & Gamble in its Hunt Valley distribution center but P&G France chooses to use “inventory early warning” Agent from SAP to accomplish its ATP check. With code, state and control localized within the Agent, little or no integration is required to launch an application (asynchronous operation). On the other hand, Agents (open architecture) can organize themselves (Multi-Agent System) to cooperate and provide integration (synchronization of processes). Parallel existence of autonomy and cooperativity makes Agents architecture attractive to all businesses.

Agent software holds the potential to level the playing field for small companies in their quest to use the same degree of sophisticated software tools used by the behemoths. In the past, they could not afford integrated package, like SAP. With Agents, process-specificity and functional integration are not mutually exclusive. Thus, it ensures that small companies with far less IT budgets can still deploy state-of-the-art Agents, may be few at a time, to eventually build that Multi-Agent System, as robust as the Enterprise System yet highly adaptable and easily updated due to its modularity. Agents may be developed by a small team with depth of process knowledge and appropriate programming expertise. Agent development may no longer resonate the “JBOPS” concept of big enterprise software or demand major capital investment. The situation could be analogous to the 1990’s mushrooming of dotcoms. Agent technology could spawn a “cottage industry” in countries like India, Romania where strength in mathematics education fuels an abundance of programming expertise that may be harnessed at low cost for Agent development.
Are we experiencing a conceptual change? Perhaps not a conceptual change but a certainly change of medium. In software terms, we may be witnessing an emerging paradigm shift from the aggregated integration of monolithic software to the distributed reductionist approach exemplified by Agents.

Since 5000 BC, Sumeria, Mesopotamia, Indus Valley, Xian had traders. The concept of marketing goods by individuals pre-dates documented history. Those traders soon learned the value of “aggregation” and thus evolved the bazaar. In conceptual terms, in the past few thousand years, we haven’t witnessed any breakthroughs. The brick-and-mortar evolution of the past century is only an extension of style, not substance, per se. Perceived migration from this “brick-&-mortar” to the 1990’s “point-&-click” was, a change of medium, not concept. Stand alone dotcoms mushroomed in the mid-1990’s but soon aggregated (Yahoo! Shopping) to “dotmalls” where customers were likely to visit in greater numbers than isolated dotcom stores. The brick-&-mortar businesses soon came up with their point-&-click stores. Brand recognition catalysed the middle road and championed the “click-and-brick” commerce that is still evolving as the new paradigm.

Although the glitter of asset-less dotcoms are considerably diminished, it is still interesting to consider eBay, for example. When it went public in 1998, it was (market) valued at nearly US$2 billion, double the US$1 billion value of the legendary auction house Sotheby’s established in 1721.

The transition to real-time adaptable business networks (ABN) will necessitate the paradigm shift, to Agents technology, which is essentially of the software medium. It will make it imperative to model business software both with agents and equations (monolithic, structured, object-oriented). The question is no longer whether to select one or the other approach but to establish a value-based mix of both and develop criteria for selecting the balanced composition of software, that can offer real-time solutions for businesses. The “balance” thus established is itself subject to dynamic changes. For experts in traditional supply chain management, the situation is analogous to a “push-pull” strategy where the push-pull boundary shifts with changing demand.

JBOPS shall have to determine and dynamically re-determine, repeatedly, this “push-pull” boundary, in its development strategy to meet the needs of future real-time enterprise solutions that will inevitably experience an increasing share of Agents scattered in the software landscape of businesses, of all sizes.

Reconsider the operation VA01. Each step in VA01 can be performed by an Agent. Each Agent could be purchased from a different vendor. What will trigger the customer’s preference for one vendor over the other? When the vendor ceases to be a vendor and is viewed as a strategic global thought leader building partnerships and capable of fostering global alliances with businesses and academia.

Ability of niche players to erode market share is not a new phenomenon. In Agents sphere, however, the sheer abundance of niche players may become a real threat given the business model that is likely to mushroom. An “Agent” software may claim a small price (US$5,000-25,000) and thus lowering the barrier to deployment by SME’s. Revenue may be based on the number of sites where Agents are deployed. One Agent at one site may cost $5000 but the same Agent if deployed in 5 sites will cost $25,000. The ability of a “cottage industry” of 10 persons to develop and market 50 agents for a gross revenue of $1 million or even more, therefore, is not an impossible scenario. For JBOPS, an additional mere $1 million in revenue is so trivial that the incentive to devote resources and foster mind-share in the Agents space may even justify raising eyebrows, if not the ire of management.

One software behemoth, is well on its way to capitalize on Agents technology. Microsoft’s Research Adaptive Systems team is working with Cycorp to equip MS Windows XPn with embedded Agents and “smart tags” using AI based Bayesian statistical probability model. Smart tags (think XML tags) seek and connect any information relevant to that word, name or description in any document. “CLIPPY” will evolve as an “Agent” who will sort your phone, e-mail, appointments, tasks according to priorities you have set or it has “learned” from your behaviour. Mobile networked agents in your Windows CE (PDA) may alert you to your boss’s message, help manage your on-line auction or send message to your girlfriend’s PDA to shift dinner time to 7:30pm to avoid conflict with your con call from 6pm to 7pm.
Growth of this emerging software medium in the form of Agents technology will complement the next generation of world wide web based services to emerge from the development of the semantic web. Web-based services in enterprise systems and execution of mainstream business process functionality will be required to accommodate a paradigm shift to keep pace with these developments, many of which will facilitate true real-time adaptability in (supply-demand) business networks. For businesses, this may be a classic “innovators dilemma” that has caused great companies with smart management to fail, often miserably and some into oblivion.

Epilogue: Adoption Curve of Agents in System Architecture

The economist Norman Poire observes that it takes about 28 years for a new technology to gain wide acceptance, which, then, fuels a period of rapid growth lasting about an additional 56 years. Almost after a century since “invention” or introduction, the innovation becomes a commodity and grows in line with fluctuations in macroeconomic forces and population. Agents-based software may follow a similar trajectory of gradual systemic acceptance followed by rapid adoption beginning about 2005.
Agents Application: A Case Study in Aircraft Maintenance (excerpt)

Based on our (CMU) experience with the RETSINA multi-Agent infrastructure, we implemented a system to solve an existing real-world problem. Specifically, we developed a multi-Agent framework that provides information retrieval and analysis in support of decision making for aircraft maintenance and repair for the US Air Force (USAF). Although the solution was developed for a specific type of aircraft, the Agents and interactions were designed to apply to a range of similar maintenance scenarios.

Aircraft maintenance in USAF is performed at different levels. The basic and intermediate levels are usually performed at the base where the aircraft is deployed, whereas periodic, comprehensive maintenance is performed at specialized depots. In both cases, maintenance is a complex process.

Initially, mechanics inspect the aircraft for discrepancies (and may also receive such information from pilots). For each discrepancy, the mechanic consults the technical manuals for a standard repair procedure. In case such a repair procedure is found and the resources (parts) are available, the mechanic proceeds with the repair. In cases where parts are not available or they are too expensive or require too much time and additional machinery for replacement or in cases where a procedure is not provided in the technical manual, a mechanic needs to consult an expert engineer. The engineer, in turn, may consult external sources of information. These include manuals, historical maintenance data and may even include consultation with experts.

Until recently, no automation was introduced to the consultation processes of this information-rich environment. Hard-copy repair manuals are used by mechanics and engineers. Thus search for relevant information may be time consuming and incomplete. Historical data (records of previous similar repairs) is scarcely used, since it is stored in paper format with no search mechanisms and usually only kept for short periods (distributed along remotely located service centers). Expert engineers may be located remotely and their advice is available by voice or fax messages, usually delayed for hours or days. All of these factors contribute to a slow, inefficient maintenance.

The inspection, consultation and repair process consists of the following steps:

[1] Aircraft arrives at a maintenance center, either at its home base or depot (depending on type of maintenance required). In both cases, the maintenance procedures must be completed within a limited time period. This period varies. Basic and intermediate maintenance must be completed within hours or a few days, whereas depot maintenance may be scheduled for several weeks (time depends on the aircraft type).

[2] Mechanics inspect the aircraft and locate discrepancies. For each discrepancy a mechanic performs the following:
   [a] browse the technical manual for repair procedures.
   [b] in case that an appropriate procedure is located, mechanic needs to verify whether it can be completed given limitations on repair time and parts availability. Mechanic may also need to consider the price of repair. For example, the technical manual may require replacing a whole wing if a crack in the wing is greater than some given threshold. This may take too long and become too expensive thereby causing delay or compromise operational activity or readiness.
   [c] if the procedure found in the technical manual can be performed, the mechanic performs it. Otherwise, the mechanic proceeds through consultation, as follows:
      [i] Mechanic fills Form 202a, standard USAF form for reporting aircraft discrepancies and requesting advice. The mechanic may attach supporting information (Fig 1). The mechanic may consult Illustrated Part Breakdown (IPB) technical manuals and possibly other experienced mechanics. Form 202a is sent for advise and authorization for non-standard repair.
An engineer, upon receipt of a Form 202a, performs the following:

[a] Uses own experience, historical repair information and technical manuals to design an appropriate repair for the discrepancy.
[b] Fills in a Form 202b, standard US Air Force form for discrepancy repair instructions. To this form the engineer may attach graphical illustration to clarify required repair procedure.
[c] Files 202a and 202b for future use as historical repair information.

When a standard repair procedure is found or on receipt of Form 202b from engineer, the mechanic performs the repair as instructed.

The current inspection, consultation and repair processes, as described above, have several problems. The multi-Agent system (MAS) implementation reported here attempts to address these problems.

The majority of the information, both historical repair information and technical manuals, is found in hard-copy format as well as hand-written pieces. Mechanics and engineers spend precious time on:

[a] Browsing manuals and searching for historical repair information.
[b] Drawing graphical discrepancy and repair illustrations.
[c] Mechanics idle, waiting for Form 202b to arrive from engineers in reply to their Form 202a
[d] Historical information is unused when stored remotely or local hard-copy is difficult to browse

Figure 1: Part of a graphical description attached to a form.
For information needs of mechanics, using manuals during inspection for diagnosis is inefficient and at times impossible due to physical constraints of the inspection environment. Scribbled information (see Figure 2) both from historical forms and current Form 202 may have limited comprehensibility. This problem intensifies due to deterioration in the quality of such information when it is transmitted via fax or photo-copied. Historical forms are kept only for two years. Time and effort spent on paperwork and filing should be used instead for diagnosis and repair. Printed information in the technical manuals (IPB) are not consistently updated.

To summarize, the problem with which we deal consists of decision support in a physically distributed and dynamically changing environment, rich in multi-modal information, where users have diverse (varying over time) information needs. This is the type of problem for which the CMU RETSINA MAS (multi-Agent system) is most appropriate.

RETSINA (REusable Task-based System of Intelligent Networked Agents) is a multi-Agent infrastructure that was developed for information gathering and integration from web-based sources and decision support tasks. It includes a distributed MAS organization, protocols for inter-Agent interaction as well as collaboration and a reusable set of software components for constructing Agents. Each Agent in RETSINA specializes in a special class of tasks. When Agents execute tasks or plan for task execution, they organize themselves to avoid processing bottlenecks and form teams to deal with dynamic changes in information, tasks, number of Agents and their capabilities.
In RETSINA, the Agents are distributed and execute on different machines. Based on models of users, Agents and tasks, the Agents decide how to decompose tasks, whether to pass them to others, what information is needed at each decision point, when to cooperate with other Agents. The Agents communicate with each other to delegate tasks, request or provide information, find information sources, filter or integrate information, negotiate to resolve inconsistencies in information and task models. The RETSINA infrastructure (see Figure 4) consists, by convention, of 3 broad types of Agents:

- **Interface Agents**
- **Task Agents**
- **Information Agents**
In RETSINA multi-Agent infrastructure, Interface Agents interact with users receiving specifications and delivering results. They acquire, model and utilize user preferences. The Interface Agents hide the underlying structural complexity of the Agent system. Main functions of an Interface Agent include:

1. collecting relevant information from the user to initiate a task
2. presenting relevant intermediate and final results
3. requesting additional information during task execution

Task Agents formulate plans and carry them out. They have knowledge of the task domain and which other Task Agents or Information Agents are relevant to performing various parts of task. In addition, Task Agents have strategies for resolving conflicts and fusing information retrieved by Information Agents. A Task Agent:

1. receives user delegated task specifications from an Interface agent
2. interprets the specifications and extracts problem solving goals
3. forms plans to satisfy these goals
4. identifies information seeking sub goals that are present in its plans
5. decomposes plans and cooperates with appropriate Task Agents or Information Agents for plan execution, monitoring and results compilation.

Figure 4: The RETSINA multi-agent infrastructure
Information agents provide intelligent access to heterogeneous collection of information sources. They have models of information resources and strategies for source selection, information access, conflict resolution and information fusion. Information Agents can actively monitor information sources.

Middle agents collect and provide information about the location, availability and capabilities of other Agents (possibly additional information about them). They may also serve as mediators, hiding the identities of either service requester Agents or service provider Agents or both. Middle Agents (Matchmakers) provide RETSINA-based MAS with openness. That is, Agents may leave and enter the system dynamically. When an Agent appears, it advertises itself with a Middle Agent. When it leaves, it informs as well. Agent disappearance as a result of Agent or network failure is detected by Middle Agent via a pinging mechanism.

The RETSINA internal Agent architecture is based on a multi-module, multi-thread design. It consists of two component types: functional units and data stores (see Figure 5).

![Figure 5: The RETSINA agent architecture.](image_url)
RETSINA agent uses four data stores:

The objective database (DB) is a dynamic data store. It stores the objectives of the Agent of which it is a component. New objectives are inserted by the communicator (from outside sources) and by the planner (from inside sources) as planning may create new objectives.

The task database is a dynamic data store. It stores tasks which were reduced to the lower level, that is, to actions. These tasks still may not be ready for execution and will wait in the task DB until the required conditions for their execution are set. When this happens, the actions are considered enabled and are scheduled for execution by the scheduler.

The task schema library is a static data store that holds tasks schemas. These are used by the planner for task instantiation.

The task reduction library is a static data store that holds reductions of tasks. These are used by the planner for task decomposition.

The functional modules, each of them an autonomous thread of control, use the data stores as follows:

The communicator receives and sends messages, parses incoming messages and creates objectives which are inserted to the objective DB. The planner performs instantiation and reduction of tasks. It takes the objectives of its Agent, decomposes them to lower level tasks and the tasks which are executable (referred to as actions) are passed forward, to be scheduled for execution.

The scheduling of enabled tasks is performed by the scheduler. It takes enabled actions from the task DB and transfers them, scheduled, in the scheduler. Activation of actions in the schedule is performed by the execution monitor thread. For each action on the schedule it creates a separate thread of control. It monitors the activity of each of these working threads. Action threads may propagate outcomes to other modules.

Modularity of RETSINA agent architecture (having no direct interfaces between its functional modules) result in code re-usability (RETSINA communicator used for multiple Agents & non-agent applications that converse with Agents). In addition, functional components can be replaced in a “plug-in” fashion.

Given its properties, we found the RETSINA infrastructure appropriate to solve the USAF maintenance problem. By developing and using an Agent architecture, we gain the following advantages:

[1] The RETSINA architecture can be used to wrap legacy software systems by equipping them with a Communicator module. Thus the resulting system remains backwardly compatible with the older systems, without restricting future software development to an obsolete model. For instance, currently the Warner Robins Air Force Base (AFB) engineers are experimenting with entering some of the data into an Access DB format, as a temporary measure while waiting for (the ITL-ALC) another system to become available. With this design, separate Info-Agents can easily be designed to accommodate both data sources. Since the maintenance personnel only interact with Interface Agents, they are shielded from internal data discontinuities.

[2] The information required by the maintenance engineers is likely to be distributed among several computers, possibly in different geographic locations. RETSINA architecture provides built-in networking support useful for developing distributed systems, in the form of the Communicator, which handles low level socket operations. The Agent Name Sever/Matchmaker allows service requesters to locate service providers. Although the current focus is on handling the repair operations described in Form 202A, which are performed locally in Warner Robins AFB, additional Agents can be added to the system to access collections of Form 00-107 (immediate repair requests) which can be filed from multiple locations. These Agents would be located on computers at the local Air Force base performing the repair and would communicate to agents at the central F-15 repair location (Warner Robins AFB).
The Warner-Robins Air Force Base is in a transitional phase of reorganizing their data and also training personnel. Rapid prototyping of a group of Agents are underway to address the current situation and slowly add to the “Agent Population” as new information sources become available electronically. Since the Interface Agent is decoupled from the Info-Agents, it is possible to replace older Info-Agents without disturbing users. Training personnel to use the system is an important part of making it operational. Changing the interface as little may facilitate the personnel training process.

Disclaimer

This summary on Agents is not an original work by the author (Shoumen Datta). I have merely written it in simple English to make it suitable for understanding by a large number of non-experts, myself included. I neither have the talent nor the training to produce the concepts and formulations described in this article. Failure to represent the ideas with clarity is entirely my fault. If you have understood and appreciated the scope of Agents, then, the credit is solely due to the brilliance of the scientists whose work I have paraphrased. I have used papers from Massachusetts Institute of Technology (MIT), Carnegie-Mellon University (CMU) and University of Michigan, Ann Arbor. The monographs by H. Van Dyke Parunak deserves special acknowledgement. Email: shoumen@mit.edu
Multi-Agent Systems in Supply Chain Triggers Innovation in Real-Time Demand Management

Linearisation of real world conditions to fit mathematical models may create lack of real-time adaptability in supply chain. A common example of this is the Bull Whip Effect that depicts wide fluctuations in supply chain. The discrete, dynamic and distributed nature of data and applications require that supply chain solutions not merely respond to requests for information but intelligently anticipate, adapt and actively support users. Agents can support a clearly discernible task or process, interact with each other in a specified environment (say, inventory management), interact with other Agents directly or via a message bus, continuously harness real-time data (from UWB, RFID tags, GPS, sensors) and share this data with other Agents to offer global true real-time adaptability in business networks through web services powered by the semantic web development.

This concept is at the heart of Multi-Agent System and it is one of the research topics at the Forum. Real-time adaptability may affect a vast array of static or pre-set business processes. It is likely that many processes may change to evolve into the paradigm shift that is leading toward the adaptable business network (ABN). In particular, real-time adaptability may evolutionize supply chain management by fostering supply chain innovation through deployment of Multi-Agent Systems. Agent-based modeling draws clues from natural behaviour of biological communities of ants, wasps, termites, birds, fishes and wolves, to name a few.

In commercial supply chain software (i2, SAP, Oracle, Manugistics) processes are defined in terms of rates and flows (consumption, production). System variables (cost, rebates, transportation time, out-of-stock) evaluate or integrate sets of algebraic equations (ODE, ordinary differential equations or PDE, partial differential equations) relating these variables to optimise for best results (best price, shortest lead time, minimal inventory). The process (EBM or equation-based modeling) assumes that these parameters are linear in nature and relevant data are available. In the real world, events are non-linear, actions are discrete and information about data is distributed.

Research at the Forum may contribute to solutions where Agents-based supply chain software may function continuously and autonomously in a particular environment, often inhabited by other Agents (Multi-Agent Systems) and processes. Continuity and autonomy indicates that Agents are able to execute processes or carry out activities in a flexible and intelligent manner that is both adaptive and responsive to changes in the environment without requiring constant human guidance, intervention or top-down control from a system operator. An Agent that functions continuously in an environment over a period of time would be able to learn from experience (patterns). In addition, Agents that inhabit an environment with other Agents (Multi-Agent Systems) are able to communicate, cooperate and are mobile (from one environment to another). The mobile, networked, autonomous, self-learning, adaptive Agent may have radically different principles compared to those that were developed for monolithic systems. Examination of naturally occurring Agent-based systems suggests design principles for the next generation of Agents. While particular circumstances may warrant deliberate exceptions, in general, the research in the Forum may align with these concepts:

1. Agents should correspond to “things” in the problem domain rather than to abstract functions.
2. Agents should be small in mass, time (able to forget) and scope (avoid global knowledge action).
3. Multi-Agent Systems should be decentralized (no single point of control/failure).
4. Agents should be neither homogeneous nor incompatible but diverse.
5. Agent communities should include a dissipative mechanism (entropy leak).
6. Agents should have ways of caching and sharing what they learn about their environment.
7. Agents should plan and execute concurrently rather than sequentially.

Computer-based modeling has largely used system dynamics based on ODE. However, a multitude of industrial and businesses, including supply chain management, are struggling to respond in real-time. Eventually this transition may emerge as real-time adaptable business network. This paradigm shift will make it imperative to model software based both with Agents and equations. The question is no longer whether to select one or the other approach but to establish a mix of both and develop criteria for selecting one or other approach, that can offer solutions. The balance is itself subject to change. For experts in supply chain management, the situation is analogous to “push-pull” strategy where the push-pull boundary may shift with changing demand.
Difference in representational focus between ABM vs EBM has consequences for how models are modularized. EBM’s represent the system as a set of equations that relate observables to one another. The basic unit of the model, the equation, typically relates observables whose values are affected by the actions of multiple individuals, so the natural modularization often crosses boundaries among individuals. ABM represents the internal behaviour of each individual. An Agent’s behaviour may depend on observables generated by other individuals, but does not directly access the representation of those individuals’ behaviours, so the natural modularization follows boundaries among individuals. This fundamental difference in model structure gives ABM a key advantage in commercial applications such as adaptable supply chain management, in two ways:

First, in an ABM, each firm has its own Agents. An Agent’s internal behaviours are not required to be visible to the rest of the system, so firms can maintain proprietary information about their internal operations. Groups of firms can conduct joint modeling exercises (Public MarketPlace) while keeping their individual Agents on their own computers, maintaining whatever controls are needed. Construction of EBM require disclosure of relationships that each firm maintains on observables so that equations can be formulated and evaluated. Distributed execution of EBM is not impossible, but does not naturally respect boundaries among the individuals (why public e-MarketPlaces failed to take-off).

Second, in many cases, simulation of a system is part of a larger project whose desired outcome is a control scheme that more or less automatically regulates the behaviour of the entire system. Agents correspond one-to-one with the individuals (firms or divisions of firms) in the system being modeled and their behaviours are analogs of the real behaviours. These two characteristics make Agents a natural locus for the application of adaptive techniques that can modify their behaviours as Agents execute, so as to control emergent behavior of the overall system. Migration from simulation model to adaptive control model is much straightforward in ABM than EBM. One can imagine a member of adaptable business network or supply chain using its simulation Agent as the basis for an automated control Agent that handles routine interactions with trading partners. It is much less likely that such a firm would submit aspects of its operation to an external “equation manager” that maintains specified relationships among observables from several firms.

ABM’s support more direct experimentation. Managers playing “what-if” games with the model can think directly in terms of familiar business processes, rather than having to translate them into equations relating observables. ABM’s are easier to translate back into practice. One purpose of “what-if” experiments is to identify improved business practices that can be implemented. If the model is expressed and modified directly in terms of behaviours, implementation of its recommendations is a matter of transcribing the modified behaviours of Agents into task descriptions for the underlying physical entities in the real world.

The disadvantages of EBM result largely from the use of averages of critical system variables over time and space. EBM assumes homogeneity among individuals but individuals in real systems are often heterogeneous. When the dynamics are non-linear, local variations from the averages can lead to significant deviations in overall system behavior. In business applications driven by ‘if-then’ decisions, non-linearity is the rule. Because ABM’s are inherently local, it is natural to let each Agent monitor the value of system variables locally, without averaging over time and space and thus without losing the local idiosyncrasies that can determine overall system behaviour.

The approach to system design and supply chain management with Agents in the software landscape is at odds with the centralized top-down tradition in current systems. The question usually arises in terms of the contrast between local and global optimization. Decision-makers fear that by turning control of a system over to locally autonomous Agents without a central decision-making body, they will lose value that could have been captured by an integrated (enterprise) global approach. The benefits of Agent-based architecture approaches vs centralized ones are conditional, not absolute. In a stable environment, a centralized approach can be optimized to out-perform the initial efforts of an opportunistic distributed system of Agents. If the distributed system has appropriate learning capabilities, it will eventually become as efficient. Market conditions are marked by rapid and unpredictable change, not stability. Change and contingency are inescapable features of the real world. The appropriate comparison is not between local and global optima but between static versus adaptable systems. Real-time adaptability is crucial to supply chain management.