Using Agents to Achieve Process Coherence

Anuj K. Jain¹, Manuel Aparicio IV², Munindar P. Singh³

Open environments such as the Internet—and even corporate intranets—make vast quantities of information available to a large number of interested parties who can access, use, and enhance it. They enable modern applications, such as manufacturing, virtual enterprises, and ubiquitous information access, which involve a number of information sources and component activities. However, without principled techniques to coordinate the various activities, the overall system behavior would remain unimpressive, and practically confine the open environments to limited, conventional approaches.

Traditional programming techniques are designed for closed environments, where the programmer has (in principle) complete knowledge of the meaning of the information and control over the disposition of the participating activities. By contrast, in open environments, a programmer has partial knowledge of and virtually no control over the behavior of the components supplied by other vendors and being executed by autonomous users. Although preserving the autonomy of participating components is crucial, unrestrained autonomy would be risky, because it may easily lead to undesirable consequences.

The agent metaphor, long in study in artificial intelligence, has recently become popular in mainstream computing, largely due to its suitability for open environments. For our purposes, the key aspects of agents are their autonomy and abilities to perceive, reason, and act in their environment, as well as to socially interact and communicate with other agents [12]. Viewed in this light, agents are the right unit of interoperability, and naturally reflect the autonomy of humans and the underlying computational systems. Put simply, our technical challenge is to manage autonomy: how to maximize freedom without letting it devolve into chaos.

Traditional techniques for information management are based on notions of data consistency. However, in many practical situations, correctness requirements cannot be defined without reference to processes. The same data state may correspond to multiple histories of actions and interactions among the participants, and only some of those histories may be deemed acceptable. For example, a delayed order is bad, but not if the customer has granted an extension to obtain a higher quality outcome. We refer to the felicity of the (desired) interactions as process coherence.

Abstractions for Composite Activities

Information management in the large involves three main concerns, which must be addressed by any approach for constructing solutions:

- data integrity and flow: correctness of data and how it is conveyed from one party to another
- organizational structure: how the various parties relate to each other
- autonomy: how the autonomy of the different parties is preserved.

Table 1 summarizes the major abstractions for programming composite activities. Database transactions are the most rigorous, and require that only correct data ever be visible [11]. This entails that outputs be released only when a transaction completes. Thus, the producer of data is restricted, but the consumer has full autonomy, unless it also is a transaction. Spheres of control (SoCs) release results early, but may undo and redo the consumers—thus, consumers have no autonomy [8]. Extended transactions release results liberally, but restrict the autonomy of their components by requiring compensating subtransactions to undo the effects of data that are invalidated [8]. Workflows ignore the integrity aspects, but capture the data flow required by specific applications [10]. They allow autonomy, but are not flexible. Spheres of commitments are discussed below.

<table>
<thead>
<tr>
<th>Support Provided (columns)</th>
<th>Integrity &amp; Autonomy</th>
<th>Organizational</th>
</tr>
</thead>
</table>

¹ Ericsson New Concepts Group, Research Triangle Park, NC.
² IBM Intelligent Agents Group, Research Triangle Park, NC.
³ Dept of Computer Science, North Carolina State University, Raleigh, NC.
Spheres of Commitment

A commitment is a relationship between a debtor, a creditor, a context, and a proposition. The debtor owes it to the creditor to make the proposition true; the context serves as a witness and as adjudicator of disputes. There is indeed a close relationship between this form of commitments and legal reasoning; some of these are studied in [S98, AI&Law]. A sphere of commitment (SoCom) is viewed conceptually as a scope within which a commitment applies; practically a SoCom is a multiagent system that serves as the context for some commitments. A SoCom is typically associated with a set of resources and authorities over them. A SoCom is modeled using a representative agent, which behaves like a group leader.

The agents can represent nonterminating computations. Thus, their results must be released prematurely, even if invalidated later. Recovery is effected by having the agents satisfy their commitments—typically, by sending notifications to other agents. The recipients autonomously process the notifications. Consequently, by enabling agents to enter into commitments, we can achieve recovery without violating the autonomy of the consumers. Our approach applies recursively in that each SoCom can itself be treated as an agent and participate in larger SoComs. We will not emphasize this aspect here.

Social actions are actions on commitments. They correspond to the following operations:
- create: instantiate, performed by debtor or context (by putting the debtor in a certain role)
- discharge: satisfy, performed by the debtor
- cancel: give up, performed by the debtor
- delegate: make another agent the debtor, performed by debtor or context
- assign: make another agent the creditor, performed by creditor or context
- release: eliminate entirely, performed by the creditor or the context

Social actions cannot be wantonly performed, but are governed by social policies. Social policies can be thought of as metacommittments, which effectively define the structure of a minisociety.

An abstract SoCom defines the structure of a SoCom using roles, instead of actual agents. Each role comes with a description of its requirements in terms of resources, capabilities, and capacity. A resource will be instantiated with an information resource. A capability is a functionality that the role presupposes. The capacity of a role corresponds to the spare capacity an agent must have in order to adopt the role. An abstract SoCom specifies the commitments associated with a role—these must be inherited by any agent who adopts a given role. An abstract SoCom also specifies some constraints on the interactions of the agents, e.g., whether an agent should initiate an interaction or wait for another agent to begin.

A concrete SoCom is obtained by naming an abstract SoCom, and binding agents to its roles. An agent may adopt more than one role in an abstract SoCom, and participate in more than one concrete SoCom concurrently. In order to adopt a role, an agent must certify that he has the requisite resources and capabilities, and can spare the needed capacity. Upon adopting a role, an agent automatically inherits all the commitments that go with that role. This is the main way in which commitments are created. The commitments associated with a role are typically metacommittments. When the agents interact, these commitments yield the base-level commitments that the agents then attempt to discharge, intuitively, through physical actions.

A SoCom manager holds the definitions of the abstract SoComs, and supports their instantiation. Agents can browse the definitions, and request creation of a concrete SoCom. The SoCom manager communicates with the other agents to determine their suitability for the specified role. If an agent offers or agrees to participate, the manager checks whether he meets the requirements. When the concrete

<table>
<thead>
<tr>
<th>Technique (rows)</th>
<th>Data Flow</th>
<th>Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional Transactions</td>
<td>Fixed</td>
<td>Consumers</td>
</tr>
<tr>
<td>Spheres of Control</td>
<td>Relaxed</td>
<td>Producers</td>
</tr>
<tr>
<td>Extended Transactions</td>
<td>Relaxed</td>
<td>Both, limited</td>
</tr>
<tr>
<td>Workflows</td>
<td>None</td>
<td>Both</td>
</tr>
<tr>
<td>Spheres of Commitment</td>
<td>Relaxed</td>
<td>Both, constrained</td>
</tr>
</tbody>
</table>

Table 1. Computational Abstractions Summarized
SoCom is created, the agents in it can perform the capabilities of their chosen roles. Upon binding to a role, an agent adopts the commitments that go with that role. The agents then act according to their commitments.

Our implementation uses the Agent Builder Environment along with the FIPA agent communication language. This approach is being applied on the SMART project (see Sidebars).

**Application in Manufacturing**

The following scenario is a simplified interaction from the SMART project. This reflects the observation that manufacturing is becoming increasingly reliant on the dynamic formation and management of extended and overlapping virtual enterprises.

Figure 1 shows a scenario where M is a manufacturer agent who procures parts for a certain product from vendors, S_p and v_s. M has to keep track of how far the schedule has progressed (a) in real time, and (b) in terms of tasks completed. M monitors the events in its environment as well as communications from the vendors. For each delivery, M determines whether it was on time, and estimates its quality to judge whether it is acceptable. Whenever a delivery fails to materialize on time, M notifies the vendor and requests their attention. For each exception, M communicates with its human user. It is important to keep the human in the loop, both for robustness and for the acceptance of the technology in the manufacturing industry, which is conservative in adoption of new technologies. If M notified U of a delay, then it notifies U when corrective action is taken.

![Figure 1 Manufacturing Agent Concrete SoCom](image)

The interesting fact about this example is not that it is conceptually simple, but that it underlies some of the key issues in manufacturing. In a nutshell, these issues are (a) enabling rather than replacing human decision-making and (b) handling the inherent exception-driven nature of a complex activity such as manufacturing. They have long proved to be a challenge in the expansion of computing into manufacturing domains.
Conclusions

Our approach seeks not data consistency directly, but a coherent state in the ongoing interactions of the participating components. This shift in focus not only facilitates automation, but is also more intuitive in being closer to some aspects of human social behavior. People cannot make irrevocable promises when they do not fully control their environments, but they can warn each other of potential problems, e.g., if an order is not going to come through.

Implementing our approach requires representing and reasoning about commitments. Although to do so can be nontrivial, the complexity ameliorated by taking advantage of the fact that the data behind the commitments would be stored anyway, e.g., in pending orders or bills that most enterprises maintain.
**Sidebar: The SMART Project**

SMART is a multiyear, multimillion dollar manufacturing applications project sponsored by the Advanced Technologies Program of the US National Institute of Standards and Technology (NIST) [1]. SMART is being run by the National Industrial Information Infrastructures Protocols (NIIP) consortium, which involves several computing and manufacturing companies [2]. SMART stands for MES-Adaptable Replicable Technology. MES stands for Manufacturing Execution Systems, which are flexible manufacturing systems.

SMART is developing agent technology for the manufacturing execution (or “make-side”) of supply-chains. The control of proprietary processes and other MES information in such real environments also includes aspects of security. SMART places agents within both spheres of commitment and virtual private networks, both working together. SMART is pioneering the serious application of a lot of advanced computing technology [3]. The specific form of SoComs used in SMART is information contracts; they are designed for the same purpose as here, which is to help autonomous entities interoperate for flexible manufacturing.


**Sidebar: Agent Communication**

Agent communication languages (ACLs) are based on speech act theory, which builds on the idea that with language you not only make statements, but also perform actions [4]. Speech act theory distinguishes three aspects of a communication: its *location* or the words, its *illocution* or meaning, and its *perlocution* or effect on a listener. The illocution is the core aspect. In artificial languages, the grammar is stylized and different tokens are used to ensure unambiguous communication. For most computing purposes, the main kinds of illocutions are assertives (informing), directives (requesting or querying), commissives (promising), permissives, prohibitives, declaratives (causing events in themselves, e.g., what the justice of the peace does in a marriage ceremony), expressives (expressing emotions and evaluations). Most existing languages are restricted to assertives and declaratives.

The Foundation for Intelligent and Physical Agents (FIPA) is an international consortium that is standardizing several aspects of agent management and interoperation. One of its major foci is an ACL standard. This ACL includes ways in which agents can inform each other or request services. FIPA compliant agents must be able to produce a *not-understood* message; in general, they must also support some of the core types of communications, e.g., for inform and request. Figure 3 shows an example of the FIPA syntax. The given message states that Buyer informs the Vendor that, in response to the current call for bids, the bids for valves at the unit price of $40. The meanings of the terms used are derived from the Auction ontology. A rigorous testable semantics remains an open challenge for ACLs [5].

```
(inform
  :sender Buyer
  :receiver Vendor
  :content (bid valves 20-units $40)
  :ontology Auction
  :in-reply-to Call-for-bids-37
)
```

Figure 3. ACL Syntax
Although we used the FIPA ACL, much of what we used could be obtained from any other ACL, e.g., the Knowledge Query and Manipulation Language (KQML) [5]. FIPA is rapidly spreading to replace KQML as the ACL of choice; it includes support for the KQML syntax to facilitate learning.


Sidebar: Agent and Multiagent Construction

There are various ways to build agents. Our approach of choice is IBM’s Agent Builder Environment [7], being used in the SMART project. ABE has an open architecture for agent construction. It includes a rule-based reasoning system along with a number of adapters, which enable external events to be raised (as interrupts), external state to be sensed (on demand), and modifications to be effected. Adapters for transport mechanisms (such as email), timers, data access mechanisms (such as for commercial DBMSs), and devices (such as pages) exist. Knowledge is specified via sets of declarative rules and facts. Adapters may be written in Java or C++. The rule engine is invoked when some external event is raised. The rules and facts are used by the rule processor to invoke the adapters to sense additional variables and to effect changes in the environment. In this manner, ABE combines rule-based and procedural specifications.

We enhance ABE with additional functionality, so that different instances of it can act as agents in a multiagent system. This functionality includes adapters for message queues, which can be used as the basis for communication among autonomous agents. Communications that arrive for an agent are queued up, and passed up to the agent one-by-one. The agent can respond to them as it sees fit.

Capabilities and resources are defined by extending corresponding classes in the SoCom package. Abstract SoComs as well as available agents and their capabilities are registered with the SoCom manager. An agent can query the manager to find the available SoComs and the agents who may play different roles in them. Once it selects a suitable SoCom, it can request other agents to help instantiate it by adopting specific roles. The manager determines suitability and if the agents acquiesce, a concrete SoCom is instantiated. Then, the agents can communicate with each other appropriate in the given application. The existence of capabilities and the requisite capacity indicates that the communications will be properly processed.


References