Using Archon, Part 3: Particle Acceleration Control

Fabien Perriollat, Paul Skarek, and Laszlo Zsolt Varga, European Laboratory for Particle Physics
Nick R. Jennings, Queen Mary and Westfield College

THE ARCHON SOFTWARE FRAMEWORK INTEGRATED TWO PREEXISTING STANDALONE SYSTEMS INTO A CONSISTENT, FLEXIBLE APPLICATION. THE INTEGRATED SYSTEMS PROVIDE USEFUL INFORMATION TO EACH OTHER AND IMPROVE EACH OTHER'S PERFORMANCE THROUGH COOPERATION.

PARTICLE ACCELERATORS, IN GENERAL, are complex systems that provide physicists with beams for their experiments—think of them as a factory that produces particle beams. The beam is formed by setting certain physical parameters (like the beam's dimensions) along its path in the accelerators and in the transfer lines between the different accelerators. The physical parameters are set by the accelerator equipment (for example, quadrupoles and bending magnets), which is controlled by the control system's computer hardware and software. The control system, which is managed by a number of operators, has six main functions:

- Automatic sequencing of the "beam factory" for different users,
- Acquisition of control values (sensor readings of beam properties and measurement devices in general),
- Control of setpoint values and equipment status,
- Analysis of alarm data obtained via surveillance programs and corresponding recovery actions,
- Execution of various test programs, and
- Archiving and restoring reference control values.

We used the Archon framework to build a distributed AI application for controlling, and diagnosing faults in, the *Proton Synchrotron*, one of CERN's (the European Laboratory for Particle Physics) particle accelerators.[1] The PS complex is the heart of CERN's accelerators and experimental facilities and also acts as an injector for the larger accelerators.

Why use DAI techniques for this application?

Accelerator control is a complex activity. Running and maintaining accelerators is an increasingly sophisticated task that requires the help and cooperation of a number of human experts—for example, the operators, the fault-finding experts, and the accelerator physics specialists. A monolithic and centralized system is no longer feasible or appropriate because of the domain's sheer size. So, CERN chose DAI as an implementation technology because it has several features that are well-suited to managing this inherent complexity. It offers a means of decomposing complex knowledge, assigning it to multiple processing entities (agents), and then recombining it, and adding value through cooperative interactions.[2]

A second motive for a DAI approach is that modern control systems for large accelerators are inherently decentralized and distributed. CERN's current control system, for instance, contains a number of heterogeneous subsystems that implement the accelerator's main functions.[3] These subsystems include a database system that contains details of the accelerator's components, an equipment access and control system, an alarm system, an operators' console system, two diagnostic systems, and a setup system.

Despite this decentralization, the control system must uphold a number of subsystem interdependencies. For example, the diagnostic systems load the structural description of the accelerator from the database. Also, the different diagnostic systems can provide useful information to one another. The setup system, software that instantiates parts of the accelerator after a failure or the whole accelerator after a shutdown period, can act as a recovery agent for the diagnostic systems. The diagnostic systems can provide details

0885-9000/96/$4.00 © 1996 IEEE

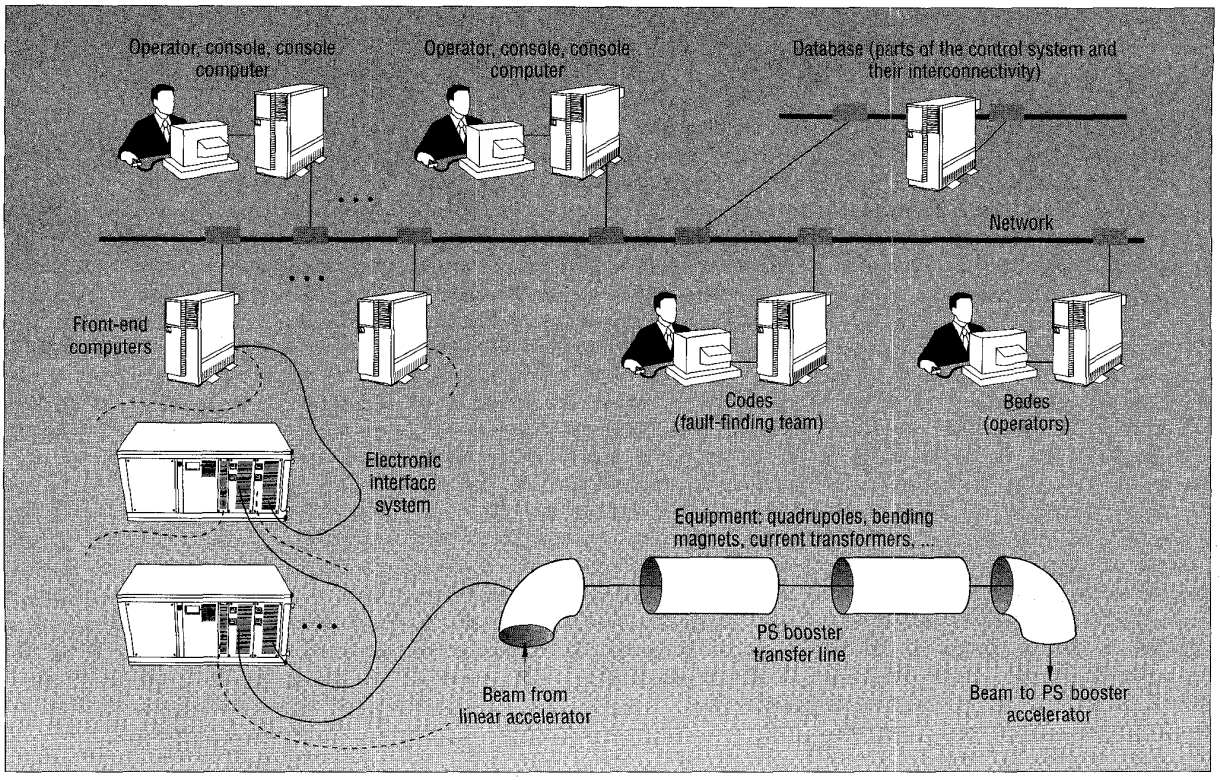

Figure 1. The PS accelerator control system and the cooperating agents.

for the setup system if the latter system fails. To effectively manage these dependencies, the separate components must work together in a coordinated, coherent fashion to meet the overarching goal of maintaining the accelerator's availability for physics experiments.

A final reason for adopting a DAI approach is that a significant amount of software existed for this application. All of this software connected to the same physical reality—the accelerator—but no facilities enabled the distinct subcomponents to interact meaningfully and intelligently. This software was tried and tested, and it represented a substantial investment. Therefore, the final delivered solution had to integrate this software and had to allow the obviously related components to interact. As well as supporting software integration, Archon offered a methodology that enabled integration of the legacy systems.

In this application, CERN based the top-down analysis on a task decomposition of the operator's job and on the principle of cooperating agents combining the individual, distributed problem-solving subsystems to work together on a common goal.[4] They analyzed the existing subsystems from the bottom up. Taking into account the available resources for development, they decided that keeping the subsystems' original size and making them cooperate would produce the best investment/improvement ratio. This analysis also revealed that the existing subsystems could provide useful information to one another and could improve each other's performance through cooperation. In particular, CERN believed that by using the Archon system, the two existing diagnostic expert systems could produce more detailed results faster.

Specification of the agents

Figure 1 shows the accelerator-control environment in which we developed the multiagent application. The operators (top-left corner) control equipment in the PS booster's transfer line through a *knob-chain*. (The PS booster is a preaccelerator, boosting the energy and the beam qualities before injecting the beam into the PS.) A knob-chain is a collection of connected hardware and software modules through which data flows. It starts at an operator's console, where the operator selects a parameter (called `OB.NAME`) to be controlled. Turning the appropriate knob on the console activates programs in the console computer and, via the computer network, the equipment driver in the front-end computers. This equipment driver controls one or more electronic interface modules connected to the real equipment (magnets, quadrupoles, and so on), where the knob-chain ends.

The two diagnostic expert systems in the accelerator control environment are *Codes* (*co*ntrol system *d*iagnosis *e*xpert *s*ystem)[5] and *Bedes* (*be*am *d*iagnosis *e*xpert *s*ystem).[6] Both connect on line to the running accelerator.

Codes finds faulty hardware and software modules in the accelerator control system when malfunctions occur. It includes model-based reasoning and on-line access to the database (see the top-right corner of Figure 1) describing the control system's parts and modules and their connectivity. It was implemented on a Symbolics Lisp machine using Intellicorp's Knowledge Engineering Environment (KEE) to write a generic shell for diagnostic reasoning.

Codes' high-level reasoning procedures are based on selecting and treating hypotheses from an *agenda*. Originally, hypotheses were created by the user via the graphics interface or by the diagnostic rules and procedures for a suspected entity. Then, either the user or Codes' diagnostic rules put the hypotheses on the agenda. In the ensuing DAI system, cooperative reasoning coming from another agent can also create hypotheses.

A Codes hypothesis is an object that comprises, among other attributes, this principal information:

- *Suspected entity*: which object in the control system is suspected (for example, a focusing quadrupole).
- *State of the entity*: what is the suspected erroneous state (for example, switched off).
- *Verification method*: how to verify or abandon this particular hypothesis.

The verification method includes both procedural data (for example, to be collected from the database or via accesses to the control system itself) and declarative data (that is, the diagnostic rules). Procedures and rules use the initial data (such as an operator's suspicions or error codes) if available, or use the results from tests being executed.

During the diagnosis, the procedural access routines and the declarative rules can create (and add to the agenda) new hypotheses that are further specializations and alternatives. These new hypotheses then become the children of the hypothesis that created them. In this way, the hypotheses form a hypothesis tree. A number of general metarules govern the selection of the next hypothesis to be treated from the agenda. For instance, if a hypothesis is on a certain module and another hypothesis is on a subpart of that module, only the subpart hypothesis is kept, because it is more detailed and therefore more promising. Or, if two hypotheses are on the same suspected entity and one of them indicates a more specific fault, the one with the more specific fault gets checked first.

Bedes helps operate the PS booster. It also helps operators set up a certain particle beam, keep stable running conditions, or change to a different kind of beam. Bedes uses the same hypothesis-centered-reasoning shell that had been developed for Codes, but the initial hypotheses that Bedes creates are monitoring hypotheses. These hypotheses correspond to discrepancies detected for beam intensities along the injection line and discrepancies in the efficiency of the injection into the PS Booster rings.

Both Bedes and Codes were conceived and implemented as stand-alone systems, so they did not interact when they were initially installed. In some instances, however, the operators of both systems cooperated orally when people from the fault-finding team were working on a particular problem. For example, if the operators failed to adjust the beam and suspected (with the help of Bedes) that a certain value could not be controlled, then they contacted the fault-finding team. The team found out (with the help of Codes) if a control module was broken or if a software table was corrupted. The control team then diverted the knob-chain to another piece of equipment.

CERN researchers expected this cooperation to be useful, because Bedes accesses the control system in its diagnosis and can provide input to Codes. Also, problems with the beam's physical properties might indicate problems in the control system, and problems in the control system might cause deviations in the beam's physical properties. If the beam is lost, either the operator made an error (the wrong setpoint value or wrong archives, both of which are in the domain of Bedes) or the control system is faulty (the domain of Codes). Either way, the symptoms on the beam are the same. Hence, the primary role of the Archon layer (AL—see Part 1, p. 64) in this implementation is to act as an arbiter that gives hints about where the real problem lies.

The cooperation method: conceptualization and implementation

A principled and well-founded DAI system required a general paradigm by which Codes and Bedes could cooperate. Our method rests on two key observations:

- An expert system's diagnostic process can be viewed as the generation and evaluation of a hypothesis tree;
- Cooperation between the expert systems can be initiated by relating hypotheses in different hypothesis trees.

The CERN application distributes the diagnostic knowledge between Bedes and Codes. For example, if Bedes finds that it cannot change a certain control value, Codes will have the knowledge to analyze the reasons for that error and to further diagnose the problem. Thus, Bedes has a hypothesis that refers to a control value with the property "cannot be changed." The knob-chain in the diagnosed domains sets this value. Therefore, Bedes' hypothesis relates to the one in Codes that refers to the knob-chain controlling that control value and that has the property "not working." *Hypothesis translation* (which we'll discuss in more detail later) relates these hypotheses, analyzing a hypothesis in one expert system and creating the appropriate hypothesis in the other.

These two observations indicate that the exchange of hypotheses provides the best means of instantiating cooperation in this application. In this case, the agents also exchange diagnostic knowledge (because of the declarative knowledge contained in the hypotheses' verification method). This exchange joins the knowledge distributed in the two expert systems—for example, one expert system can supply the suspected entity and the state of the suspected entity, and the other can supply the verification method.

As we stated previously, both Bedes' and Codes' agendas indicated the state of the diagnostic process. The expert systems changed their agendas after each complete inference cycle. In both cases, an inference cycle involves selecting the most highly rated hypothesis (the one at the front of the agenda) and evaluating it. This evaluation can result in new hypotheses being inserted into the agenda, old ones being removed, or existing ones being modified (having their ratings changed). Therefore, its completion represents the most natural point for hypothesis interchange. Given this agenda-based encapsulation of each agent's diagnosis state, we decided that the easiest way to realize the effects of hypothesis exchange would be for the Archon layer to manipulate the agenda. This means the AL can add, remove, or modify hypotheses—just like its underlying intelligent system (IS—see Part 1, p. 65)—but that it need not be involved in the domain-dependent details of hypothesis evaluation.

Exchange of hypotheses and manipulations of the agenda by the AL were the crucial points that opened up Codes and Bedes for cooperation. This method's advantage is that the expert systems and their respective ALs can easily connect. The agenda provides a clean interface, because we do not have to modify the basic functioning of the expert systems and because we can easily separate the agenda-manipulation tasks from the existing systems. In this way, the AL acts as a broker of hypotheses and initiator for changes in their ratings, while the underlying IS acts as a hypothesis evaluator whose focus of attention can be set through both local and cooperative know-how.

The implementation of this generic cooperation method required[7]

- opening the expert systems so that the internal tasks become visible to the AL as IS tasks,

```
AAM CODES
   (HYPO NEW_HYPOS)          ; Unconstrainted interests
```

Figure 2. A portion of Bedes' acquaintance model of Codes.

```
TRIGGER T_InferenceCycle
 (AGENDA)                                    ; input data
 [ IF (NOT(CMP(AGENDA,"()")) )               ; condition part
   EXEC(InferenceCycle); ]                   ; action part
```

Figure 3. Trigger that starts Codes' `InferenceCycle` behavior.

- implementing the new cooperation-specific tasks,
- implementing the translation tasks that relate hypotheses in different trees to one another, and
- instantiating the AL so that the expert systems can exhibit the desired cooperative behavior.

To open the expert systems, we needed to identify each of their main activities and encapsulate the activities as monitoring units (MUs—see Part 1, p. 66) in the AL (because of Bedes' and Codes' similar implementation structure, the same analysis applied to both). We identified these activities: **INITIALIZE**, which sets up the expert system; **EVALUATE_NEXT_HYPOTHESIS**, which evaluates a single hypothesis; **REARRANGE_AGENDA**, which applies metarules and rearranges the agenda according to the priority ratings; and **INJECT_HYPOTHESIS**, which places a new hypothesis onto the agenda. The modular design of the expert systems meant that making these activities available as IS tasks required no significant effort. By directly starting these tasks, the AL can execute inference cycles in its underlying expert system. By starting the **INJECT_HYPOTHESIS** task, the AL can also insert into the agenda hypotheses coming from cooperation.

To satisfy the additional needs coming from multiple-agent interactions, we had to develop new cooperation-specific tasks that fell into two categories. First, we needed simple tasks that make some previously internal information visible to the AL. One such task is **GET_AGENDA** , which returns the current agenda. This task helps find out whether the agenda is empty. The other is **HYPO_IN_AGENDA**, a Boolean task that returns "true" if the given hypothesis, or any of its children, can be found on the agenda. This task helps determine whether the diagnosis concerning a certain hypothesis is finished.

Second, we needed tasks that exploit the advantages of cooperation. For example, **DECREASE_ATTENTION** and **INCREASE_ATTENTION** focus the expert system's attention; they correspondingly decrease and increase the priority rating of a given hypothesis and all its children on the agenda. The final required task is **CHECK_CONFLICT**, which checks whether the results provided by Bedes and Codes contradict one another. This task encodes the fact that Codes has better knowledge of the control system and thus is always more reliable. The tasks in this second category are somewhat more complex than those in the first. Nevertheless, they are reasonably simple, and they provide the expert systems with strategic information coming from cooperation.

We implemented the translation tasks by relating the beam's physical properties and control elements in the control system. The tasks are relatively straightforward and involve relating suspected entity names to each other so that each expert system's knowledge base can refer to them. The first translation task, **INJECT_HYPOS_AND_TRANSLATE**, converts a hypothesis received from another agent. For example, when Codes receives a hypothesis from Bedes, it must create a hypothesis for the part of the control system that controls the physical property or control value suspected by Bedes' hypothesis. The second translation task, **BACK_TRANSLATE**, translates back those hypotheses that are children of a hypothesis received from another agent. For example, when Codes evaluates the children of the hypothesis created by **INJECT_HYPOS_AND_TRANSLATE**, it has to relate the hypothesis to the root of the tree and report the result back to the sender of the original hypothesis.

To instantiate the AL, we need to populate the acquaintance and self models (see Part 1, p. 67) and define the skills and behaviors (see Part 1, p. 67). In this application, the acquaintance models primarily describe which hypotheses should be sent to which acquaintances in which circumstances. Three cases must be considered. The first is when Bedes creates new hypotheses. Bedes should always send them to Codes so that it can verify those parts of the control system that are referenced in these hypotheses. The second case is when Bedes evaluates a hypothesis. This evaluation can contain useful information for Codes. For instance, confirmation of the hypothesis might indicate to Codes that the control-system modules related to the object referred to in Bedes' hypothesis might be faulty. Nonconfirmation of the hypothesis might indicate the opposite. The third case is when Codes evaluates a hypothesis that is a child of a hypothesis received from Bedes. Codes should report the result back to Bedes, because if Bedes is still working on the hypothesis and its consequences, Codes' result can be useful (as we'll discuss in the next section).

Figure 2 shows a portion of Bedes' acquaintance model of Codes. After each inference cycle (particularly after the **EVALUATE_NEXT_HYPOTHESIS** task), Bedes should send the hypothesis that has just been evaluated (**HYPO**) and those just created (**NEW_HYPOS**) to Codes in all cases (the missing condition signifies "in all cases").

In this application, the self models mainly describe the underlying IS's local skills and behaviors. The skills that control the agent's main domain-level problem-solving activities are **StartUp** and **InferenceCycle**. **StartUp** executes the initialization when the agent starts. The planning and coordination module (PCM—see Part 1, p. 68) calls **InferenceCycle** during the diagnosis to execute one inference cycle; this skill involves executing the **EVALUATE_NEXT_HYPOTHESIS** task. It triggers when the **AGENDA** is not empty and there are hypotheses to evaluate (see Figure 3).

The skills that control the cooperation-specific activities are **AttentionTracking** and **InjectHypos**. **AttentionTracking** manages the focusing of an agent's problem solving in light of the information received from its acquaintances; we'll explain it in detail in the next section. **InjectHypos** triggers when the agent receives data of the **NEW_HYPOS** type that contains the **CREATED.BY.AGENT BEDES** string (see Figure 4). The second condition prevents this behavior from triggering in the agent that created the hypothesis. **InjectHypos** translates the hypotheses from cooperative actions and puts them into

```
TRIGGER T_Inject
  (NEW_HYPOS)                                    ; input data
  [ IF (CONTAIN(NEW_HYPOS,                       ; condition
          "CREATED.BY.AGENT \"BEDES\"") )
    EXEC(InjectHypos); ]                         ; action
```

Figure 4. Trigger for Codes' `InjectHypos` behavior.

```
BHVR InjectHypos
  (NEW_HYPOS)                                    ; mandatory input
  ()                                             ; optional input
  (AGENDA)                                       ; results
  ()                                             ; child behaviors
  (                                              ; plan
    mInjectHypos "seize.AGENDA"
      ( (mGetAgenda ""
        ( (END "release.AGENDA") )) )) )
```

Figure 5. `InjectHypos` behavior.

the agenda of the underlying IS (see Figure 5). First, the **InjectHypos** behavior locks the agenda (using the **AGENDA** semaphore) to avoid other behaviors changing it while this behavior is active. Next, **InjectHypos** places the hypothesis on the agenda (using the **mInjectHypos** MU, which is associated with the IS task **INJECT_HYPOTHESIS**). Then, it returns the agenda's status (using the **mGetAgenda** MU, which is associated with the cooperation-specific task **GET_AGENDA**). Finally, it releases its lock on the **AGENDA**. This behavior results in the data item **AGENDA** (that is, the agenda's status).

The implementation of the generic cooperation method had three characteristics. First, the existing systems required only small modifications (after the careful design, we carried out the modifications in a few days). Second, the AL included all the cooperative features. Third, the implementation allowed stand-alone operation of the expert systems.

A cooperative scenario

We'll now describe in detail this application's main cooperative scenario. This scenario shows how the agents can focus each other's problem-solving activities by sharing relevant and timely information (hypotheses). It has two phases. First, Bedes sends unevaluated hypotheses to Codes, which starts Codes' diagnosis activities. Even this very modest information exchange represents an improvement over the original stand-alone system, in that Codes does not have to continuously monitor the accelerator. (The accelerator can now start on demand by the receipt of information from Bedes.) Second, Bedes sends evaluated hypotheses to Codes, which enables Codes to concentrate its efforts on a further-reduced area of the problem space.

Initially, Codes is inactive and has an empty agenda. When Bedes' **StartUp** behavior executes, Bedes automatically inserts onto its agenda two general monitoring hypotheses: **EFFICIENCY** (a suspicion about bad injection efficiency) and **INJ.TRA** (a suspicion about wrong particle trajectories). Because Bedes' agenda is not empty, its AL starts the **InferenceCycle** behavior (the **T_InferenceCycle** trigger is satisfied). Bedes evaluates its first hypothesis (B1 in Figure 6). As part of the evaluation, it acquires the beam intensities via the beam-current transformers along the injection part of the PS booster transfer line. It finds that the intensity is too low (say, only 70% of the incoming beam's intensity). Running the rules to justify this, Bedes creates three new hypotheses: B2 (a focusing quadrupole is not correctly set), B3 (a measurement screen forgotten in the beam line is a beam obstacle), and B4 (the setpoint value for a bending magnet is wrong). These hypotheses represent a suspected fault for some elements along the beam line that could explain the intensity drop.

The **InferenceCycle** behavior produces the result **NEW_HYPOS**, from its **EVALUATE_NEXT_HYPOTHESIS** task, which Bedes sends to Codes as unrequested information because of its acquaintance model definition (see Figure 2). The reason for sending such preliminary hypotheses is that Codes might be able to find the result more quickly or that it might reveal more details about the fault. Upon receipt of the hypotheses, Codes' **INJECT_HYPOS_AND_TRANSLATE** activity fires. This converts B2, B3, and B4 into their corresponding hypotheses (C1, C2, and C3—see Figure 6) about knob-chains, with which Codes can work. For example, for B2, Codes should suspect the subelements in the control chain for this quadrupole (for example, a wrong setpoint value, a faulty electronic interface module, or an uninitialized data table in the equipment driver), represented by the hypotheses C4, C5, and C6. When the translation is complete, Codes's AL invokes its **InjectHypos** behavior, which adds the new hypotheses to its agenda. Because Codes now has a nonempty agenda, its **T_InferenceCycle**

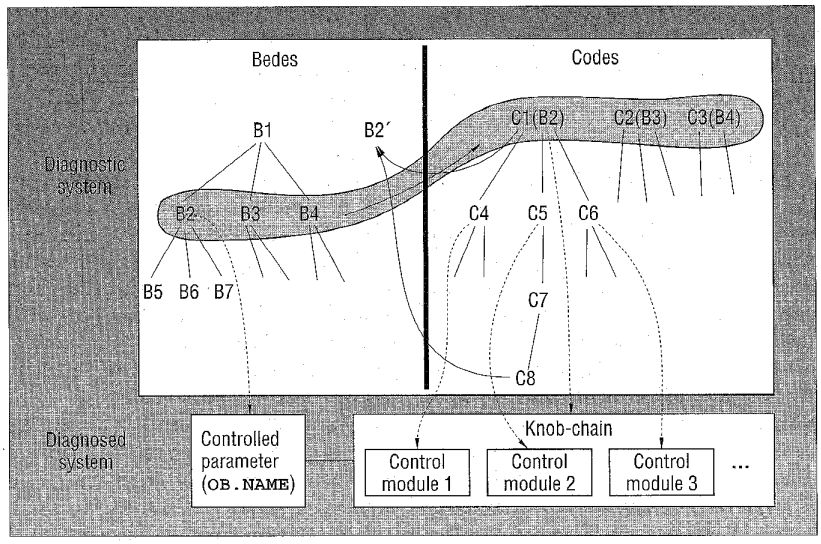

Figure 6. Cooperation through hypothesis interchange.

```
TRIGGER T_HYPO
  (HYPO)  ; input data
  [IF (CONTAIN(HYPO,                                    ; condition
         "CREATED.BY.AGENT \"BEDES\"") )
   EXEC(AttentionTracking); ]                           ; action
```

Figure 7. Trigger for Codes' `AttentionTracking` behavior.

```
BHVR AttentionTracking
  (HYPO)                                  ; mandatory input
  ()                                      ; optional input
  ()                                      ; results
  ()                                      ; child behaviors
  (                                       ; plan
    (mHypoInAgenda "seize.AGENDA"
     ( (mDecreaseAttention
         "contain.(Hypo.\"\\\"NOT.CONFIRMED\\\"\") "
          ( (END "release.AGENDA") ))
       (mIncreaseAttention
         "contain.(Hypo.\"\\\"CONFIRMED\\\"\") "
          ( (END "release.AGENDA") ))
       (mCheckConflict ""
          ( (END "release.AGENDA") )) )) )
```

Figure 8. `AttentionTracking` behavior.

trigger is satisfied and the `InferenceCycle` behavior starts. While Codes' agenda remains nonempty, the PCM continually invokes the `InferenceCycle` behavior to process each hypothesis.

The scenario's second phase starts when Bedes evaluates its hypotheses (B2, B3, or B4) and determines that one of them is confirmed (or is not confirmed). Its `InferenceCycle` behavior returns the evaluated hypothesis as a separate data unit (called `HYPO`, as opposed to `NEW_HYPOS`, which are the unevaluated hypotheses). Based on Bedes' model of Codes (see Figure 2), the behavior sends out the updated information as unrequested data.

Receipt of the evaluated hypotheses by Codes means that its `T_HYPO` trigger (see Figure 7) is satisfied and that its `AttentionTracking` skill (see Figure 8) is invoked. This behavior uses this additional information to direct its own problem solving. Codes will already have received `HYPO` from Bedes as an unevaluated hypothesis (phase 1 of this scenario) and will have injected it into its agenda, maybe started to work on it, and perhaps even finished its own evaluation.

Based on the additional information in `HYPO` and Codes' current processing status for its own version of `HYPO`, Codes' `AttentionTracking` behavior might decrease or increase attention for certain areas of the problem space or check for a conflict situation. First, the `AttentionTracking` behavior locks the agenda to stop other behaviors from changing the agenda. Then, it checks if `HYPO` (or one of its children) is currently in the agenda. It achieves this through the `mHypoInAgenda` MU, which is attached to the `HYPO_IN_AGENDA` cooperation-specific task.

`HYPO`'s presence in the agenda indicates whether Codes' diagnosis related to `HYPO` is ongoing or has been completed. If the diagnosis is ongoing, then the rating of those hypotheses that are `HYPO`'s children must increase or decrease, depending on the status of Bedes' evaluation. If Bedes has confirmed `HYPO`, `HYPO` shows promise, and its children's ratings should increase. (Codes achieves this by the `mIncreaseAttention` MU, which is attached to the `INCREASE_ATTENTION` task.) If Bedes cannot confirm `HYPO`, then `HYPO` shows little promise, so its children's ratings should decrease. (Codes achieves this by the `mDecreaseAttention` MU, which is attached to the `DECREASE_ATTENTION` task.) Codes calls these MUs selectively, using constraints between them.

If Codes has finished its diagnosis related to `HYPO`, then it must check if the new information from Bedes contradicts its own findings. (Codes achieves this by the `mCheckConflict` MU, which is associated with the `CHECK_CONFLICT` task.)

Although this scenario concentrates on Bedes focusing Codes' reasoning, the reverse can happen: Codes sends evaluated hypotheses to Bedes. Sending hypotheses in this direction, however, adds complexity; Codes must back-translate the hypothesis before sending it. To achieve back-translation, Codes' `InferenceCycle` behavior produces an additional Boolean variable (called `HYPO_TREE_FINISHED`). This variable indicates whether the whole hypothesis tree (to which the hypothesis evaluated in the inference cycle belongs) has been evaluated. The `BACK_TRANSLATE` behavior triggers if Codes has evaluated a hypothesis that is not a root hypothesis of a knob-chain (the evaluation of the root hypothesis cannot result in finding the fault). The behavior also triggers if Codes has evaluated a whole hypothesis tree (`HYPO_TREE_FINISHED` has the value "true") and the fault is not confirmed. The `BACK_TRANSLATE` behavior results in a hypothesis that has the same form as Bedes' hypotheses. Codes sends this result to Bedes because Codes' acquaintance model specifies that Bedes is interested in back-translated hypotheses. After receiving this information, Bedes can shift its problem solving away from the areas that are connected to the back-translated hypotheses, because these are unlikely to be the fault's cause.

THE DAI SYSTEM ENHANCES control-room operation in five ways. First, it can give more details and more accurate results than do stand-alone systems, so information can be presented to the system's operators more coherently. In stand-alone operation, Bedes can tell that a certain control value cannot be set, and then the operators can ask Codes why the control value does not work. The DAI system, on the other hand, can immediately present the information that the beam's efficiency decreased or that the beam is lost because a control module is stopped and a control value cannot be set.

Second, the DAI system produces the results more quickly. This speedup has two factors:

- The information transfer between the expert systems is now direct (no need to wait for the result from one expert system and enter it into the other);
- The agents can exchange partial results, so they can home in on more promising areas more quickly.[8]

Third, the intelligent exchange of hypotheses improves the focus of diagnosis. (A previously uninformed search becomes better informed, and those hypotheses that are likely to lead to the final fault get higher ratings and are evaluated earlier).

General lessons from Archon

The Archon approach was influenced greatly by the need to incorporate preexisting industrial control software into a multiagent community. However, we feel that the construction of most complex systems will involve cooperative interworking, heterogeneity, semiautonomy, and loose coupling (even if they are built entirely from scratch and are not deliberately conceived as DAI systems). This belief rests on three observations. First, most large organizations, where the majority of complex systems reside, have departmental structures that must be observed, but the individuals in these structures often must work together coherently. Second, the components (including both the humans and the software) in an organization, and even in a department, are likely to be heterogeneous, simply because each one must be based on a different model to be effective. Finally, complexity is best handled by devolving responsibility for decisions to the level at which the actions are performed (hence, the problem solvers will be loosely coupled and semiautonomous).

The two applications we've described substantiate these observations. The domain systems that already existed were each fairly complex and had been designed to encompass those aspects of the domain that could be expressed in a coherent model. They were designed in this manner because conventional wisdom dictates that when building a system using a particular modeling approach, you should include everything known about the system's world that can be expressed in that model.

However, our experience with the electricity transportation management application shows that once a cooperative framework combines several such conventional systems, completeness and comprehensiveness are no longer the key criteria for allocating agents. In a multiagent system, questions of the overall system's efficiency are likely to be paramount. This, in turn, might dictate that an agent is identified with the smallest possible coherent and autonomous entity. Consequently, the system might have a large number of such agents, which might, in turn, affect the overall system's performance. The Archon experience does not yet extend to systems with many (hundreds) of agents. However, in such situations, the designer might need to consider if some of the smaller agents must be coalesced.

Our experience with the particle accelerator control application is also of interest. Using Archon to allow cooperation between two existing systems made it possible to extend the maintenance and control system to other parts of the accelerator complex in an affordable, maintainable, and incremental manner.

Our work with both these systems has also shown us that

- Speed of operation is a factor even where the real-time requirements of the underlying processes are longer than milliseconds (because many concurrent processes are active and their interactions are cumulative);
- Passing of intermediate results (or progress reporting) effectively increases system parallelism;
- Data that is exchanged between agents ought to have a degree of persistence, so that troubleshooting can take place and audit records can be maintained;
- Significant improvements in the overall application can result from relatively straightforward cooperative interactions.

Finally, it is important to work out a clear and detailed DAI methodology, especially in view of the above discussion about agent granularity. Archon's informal hybrid approach is an important first step in this direction, but it needs to be made more rigorous and have an even clearer link to the software development process.

Fourth, the subsystems can still be separate, so the relevant different groups of experts can more easily develop and maintain them (this is why Bedes and Codes were developed originally as stand-alone systems).

Finally, the DAI system's level of integration means that users can obtain a broader and more unified view of the particle accelerator's operation and that the system's results can be presented coherently.

At present, the expertise of Bedes and Codes only covers certain parts of the accelerator complex. For maintainability, we cannot extend them to cover the whole complex. Therefore, we need to develop a number of new intelligent systems and incorporate them into the DAI system. Particularly important is the development of a setup system[9] and a timing diagnostic system. The setup system would initialize the control system and take the necessary recovery actions after a fault has been detected and diagnosed. The diagnostic system would complement the areas of expertise of Codes and Bedes by embodying diagnostic knowledge about timing problems—a common cause of faults in accelerator control. This diagnostic system typifies an incremental approach that we will adopt to extend diagnosis and recovery to the whole accelerator complex. Namely, we plan to clone new Archon agents that differ from one another only in that their domain-level knowledge bases each address a different aspect of accelerator diagnosis.

References

1. F. Perriollat, P. Skarek, and L.Z. Varga, "Report on the CERN Application Study," tech. report, Archon Public Deliverable 1060, Atlas Elektronik, Bremen, Germany, 1993.
2. F. Perriollat and P. Skarek, "Applications of Distributed Artificial Intelligence for Accelerator Control," *Proc. 11th IASTED Int'l Conf. Applied Informatics*, Int'l Assoc. of Science and Technology, Calgary, Alta., Canada, 1993, pp. 129–130.
3. F. Perriollat and C. Serre, "The New CERN PS Control System—Overview and Status," *Proc. Int'l Conf. Accelerator and Large Experimental Physics Control Systems*, special issue of *Nuclear Instruments & Methods in Physics Research. Section A, Accelerators, Spectrometers, Detectors, and Associated Equipment*, Vol. A347, 1993, pp. 86–90.
4. J. Fuchs et al., "Distributed Cooperative Architecture for Accelerator Operation," *Proc. Second Int'l Workshop Software Engineering, AI and Expert Systems for High Energy and Nuclear Physics*, World Scientific, Singapore, 1992, pp. 507–515.
5. E. Malandain, S. Pasinelli, and P. Skarek, "A Fault Diagnosis Expert System for the CERN PS," *Proc. Europhysics Conf. Control Systems for Experimental Physics*, European Center for Nuclear Research (CERN), Geneva, 1987, pp. 217–220.
6. E. Malandain, "An Expert System in the Accelerator Domain," *Proc. First Int'l Workshop on Software Engineering, AI and Expert Systems for High Energy and Nuclear Physics*, Lyon Villurbanne, France, 1990.
7. N.R. Jennings et al., "Transforming Standalone Expert Systems into a Community of Cooperating Agents," *Int'l J. Engineering Applications of Artificial Intelligence*, Vol. 6, No. 4, Aug. 1993, pp. 317–331.
8. V.R. Lesser, "A Retrospective View of FA/C Distributed Problem Solving," *IEEE Trans. Systems, Man and Cybernetics*, Vol. 21, No. 6, Nov.–Dec. 1991, pp. 1347–1362.
9. G. Daems et al., "A Knowledge Based Control Method: Application to Accelerator Equipment Setup," *Proc. Int'l Conf. Accelerator and Large Experimental Physics Control Systems*, special issue of *Nuclear Instruments & Methods in Physics Research. Section A, Accelerators, Spectrometers, Detectors, and Associated Equipment*, Vol. A347, 1993, pp. 325–328.

The authors' biographies are on p. 70.