The SPLENDORS Real Time Portfolio Management System (Extended Abstract)

Tushar M. Patel*
IBM
42FA/235
Neighborhood Road
Kingston NY 12401
914-385-8638
tushar@kgnvmx.vnet.ibm.com

Gail E. Kaiser
Columbia University
Department of Computer Science
500 West 120th Street
New York, NY 10027
212-854-3856
kaiser@cs.columbia.edu

14 April 1991

1. Introduction

We are concerned with knowledge-based applications that attempt to exploit changing market conditions to optimize the value of individual portfolios. The archetypical application is an on-line portfolio management system where a prices database is updated by a real time feed, and the user workstations repeatedly query the database as part of managing their portfolios. The massive amount of *rapidly changing data*, i.e., the prices, places high performance demands on such systems. Consistency requirements pose additional difficulties. In particular, if the analysis of the market conditions is performed using an outdated price, and a decision is made on this basis but is then executed using a substantially different current price, the results can be disastrous.

Profit represents an attempt to overcome these difficulties, by providing programming language support for developing applications with such soft real time requirements. Profit is a *coordination language* [1] built on top of the C computation language. The low-level analytic algorithms are implemented using conventional C statements and data structures, and Profit adds the communication, synchronization, multi-threading and timing facilities. Profit is intended for use by sophisticated programmers to construct distributed financial services applications.

SPLENDORS is a sample real time portfolio management application built using Profit. SPLENDORS provides a packaged library of common components as well as facilities for populating additional libraries. The libraries make it possible for financial analysts and brokers, usually non-programmers, to construct their own portfolio managers. The end-users carry out a simple, menu-based parameterization of selected components in order to incorporate their specific expertise and investment philosophies. The resulting portfolio managers are then added to the SPLENDORS system, without the need to bring down the application for recompilation and restart.

2. Background on Profit

The Profit language has been motivated by the real time portfolio management application. This involves real time monitoring of market conditions, execution of strategies using uptodate market conditions information, and updating portfolios on a real time basis to take advantage of the market conditions. This section contains a brief description of Profit. For a more detailed description, see [3, 4].

---

*The work by Tushar Patel is being carried out as a part of his MS Thesis at Columbia University.

There are three main concepts in Profit — facets, objects and processes (see figure 2-1):

- A *facet* is a minimal unit of data and control. It is analogous to a frame in that it made up of slots, each of which could contain either data or code. However, in addition to such slots, a facet also maintains a queue of requests for access to its slots and schedules the requests. Only one request may be granted at a time. Each facet contains a special slot known as the active slot, which points to the slot being currently accessed. Slots have priorities associated with them, which are used in scheduling the requests.
- An *object* is a collection of one or more facets. It represents a coherent piece of knowledge. Additionally, it binds relationships between facets in the same object and contains an interface for understanding messages received from other objects. A Profit object is illustrated in figure 2-2
- A collection of facets executes in an address space, an operating system *process*. Different groups of facets can reside in different processes, possibly on different machines, such as a prices database on a mainframe and the portfolio managers on analysts' workstations. Processes and objects are orthogonal concepts: processes do not contain objects and objects do not contain processes, but both can contain facets.

Facets may be shared among multiple objects. This is a key characteristic of Profit and is exploited throughout SPLENDORS. It makes it possible to have a single copy of rapidly changing data. For example, there is only one facet representing the price of a security even though the security may be contained in multiple portfolios (represented as multiple objects). This in turn makes it easier to maintain consistency among various objects referring to the same piece of shared data.

Each facet belongs to one and only one process. There are no copies of facets. Different facets in the same object may reside in different processes, so an object may be divided among multiple processes and there may be parts of several different objects in the same process.

Profit programs execute by sending messages between objects. When an object receives a message, it looks up the name of the message in an entry table to see which of its facets knows how to respond to this message, and then queues this message at that facet. All such messages correspond to requests to get or put a data slot or to get (execute) a code slot in a facet. A facet may handle a message by doing some processing and then sending a response back to the sender of the message, or it may itself request help from other facets in the same object — or even from other objects. So a facet can also send messages to other facets and to objects, specifically according to the relations specified in its enclosing object(s).

Data slots have get and put priorities associated with them, while code slots only have get priorities. When the current request is finished, the facet selects the next request in the queue according to which has the highest priority (rather than according to which arrived first). Both get and put requests may run either synchronously or asynchronously. A synchronous request results in the sender's own work being blocked until a response is received. For asynchronous requests, no such blocking is performed; the sender has essentially farmed out part of the processing and continues on with its portion. Constructs to block on a real time basis (viz. `everytime`, `sleep`, `start_timer`) are also provided. Different facets may run concurrently within the same or different processes. The Profit kernel manages concurrently executing facets and performs round robin scheduling among the active facets within a process. Within a facet, however, requests are not processed concurrently.

3. SPLENDORS Overview

SPLENDORS is a portfolio management application built using the Profit language. The goals of SPLENDORS are to provide a system that manages multiple portfolios consisting of stocks and options on a real time basis and to provide a means for non-programmer financial experts to build their own portfolio managers from a predefined knowledgebase. The overall architecture of SPLENDORS is illustrated in

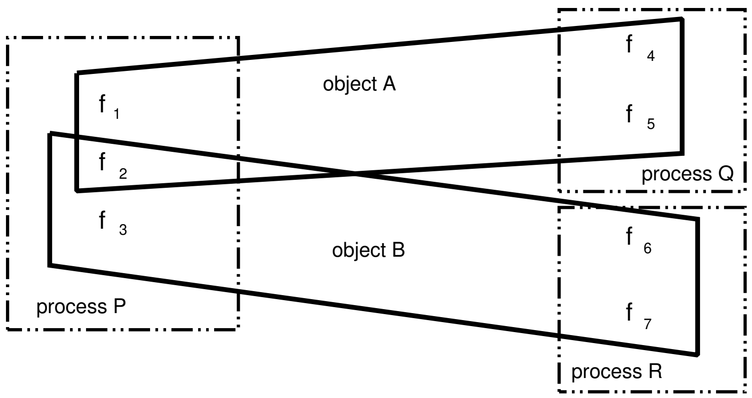

Figure 2-1: Profit facets, objects and processes

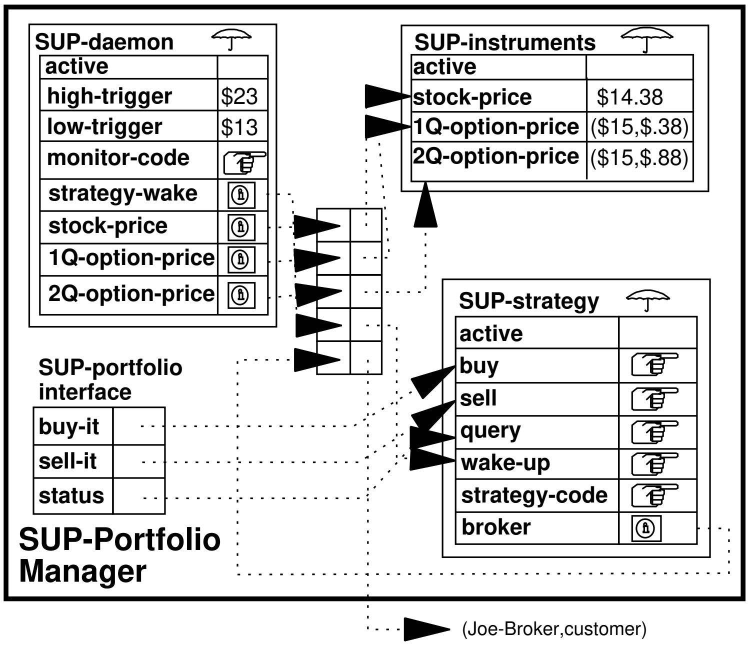

Figure 2-2: Profit object

figure 3-1. The major components of SPLENDORS are agents, strategies, prices, the knowledgebase library (or libraries) and the housekeeping object.

The idea is to implement a single prices object that contains the current prices of all the securities. Each such security is represented by a facet. Each portfolio is represented by a separate object made up of several facets. Some of these represent current prices of the securities that make up the portfolio. In fact these facets would be shared among the prices object and all portfolio objects containing that security.

Associated with each such securities facet in a portfolio, is an *agent* that monitors the price fluctuations for that stock or option. Since agents may be constructed per security per object, two portfolios containing the same security may have different criteria for monitoring the price fluctuations. This is described in greater detail in the next section.

In addition to prices and agents is the notion of a portfolio-wide strategy. Such a strategy represents the

Figure 3-1: SPLENDORS architecture

investment philosophy of the investor or fund owning the portfolio. Each portfolio contains at least one such strategy. Strategies are described in greater detail a little later.

The housekeeping object is the overall SPLENDORS-wide object that handles initialization and termination. Once initialized, SPLENDORS basically runs ‘‘on its own’’ as the prices fluctuate, activating appropriate agents and strategies.

3.1. Prices

The prices object manages the changes in the prices on a real time basis. Its design allows it to be based on an actual real time feed.

The price for each security is represented by a unique facet. Since Profit facets may be shared, this makes it possible to share that facet among the prices object and all the portfolio objects that contain that security. The sharing means that only one copy of the current price for that security needs to be maintained. Any access to a price facets by any other object anywhere in SPLENDORS transparently results in a message to the prices object.

The slots in a facet may have get and put priorities associated with them. Using a higher value for the put priority than the get priority for the slots representing a price ensures that any new prices get posted before any pending get request is serviced, thus always supplying the latest value of the price.

The management of the prices object is done through a single routine running asynchronously at a very high priority. Its job is to simply interpret the incoming feed, determine which facet to update (using a hash function on the company symbol) and carry out the update. (i.e., send a put request to that facet)

3.2. Agents

A real time portfolio manager needs to be aware of changes in the market (mainly prices). The traditional approaches include active values or polling. In the active values approach, messages describing the changes are automatically sent to all interested portfolio managers. The polling approach requires each interested portfolio manager to explicitly check the current value of the price over and over again.

In a real time portfolio management system, there are many changes in prices. Using the active values approach would mean flooding the system with a large number of messages even though most of these price changes would not be significant (since prices tend to fluctuate in small increments). Polling would solve this problem, since each interested portfolio could decide how often to poll, but would mean that the programming of portfolio-wide strategies would have to be interspersed with polling code. This gets complicated and error-prone when the number of securities becomes large and polling of multiple securities is needed.

Agents provide a solution to these problems using an intermediate approach. An agent contains knowledge that enables it to use the appropriate polling criteria. The polling interval can be different for different agents monitoring the same security. Agents also contain one or more rules. The agent notifies the portfolio manager only if such a rule is matched. This filters out uninteresting changes and ensures that only potentially interesting knowledge regarding price fluctuations is passed on to the portfolio manager.

Agents can contain arbitrary knowledge and they may carry out arbitrary computations. In fact, the knowledge in the agents could be updated dynamically. For example, the thresholds used by certain rules in agents could be updated as the market fluctuates to capture changes in trends [5].

Agents are implemented in SPLENDORS using Profit facets. Agents are conceptually associated with securities in portfolios making it possible for different portfolios containing the same security to have different agents. However, locating the agents in the same process containing the prices object minimizes network traffic and cost (because uninteresting changes do not generate any messages between an agent and its portfolio manager).

Just as there may be cases when different agents are needed to monitor the same security in multiple objects, there may also be a need for a agent to be shared among different portfolios. For example, multiple portfolios managed by the same brokerage firm might use the same polling criteria and rules for determining significance of changes. Since agents are represented by facets, they may be shared among multiple portfolio objects.

3.3. Strategies

Each portfolio includes at least one strategy that represents the investment goals of the portfolio. This knowledge is used when the portfolio manager receives notifications from its agents about changes that might be potentially interesting, to decide upon the action to take. For example, the strategy of some investor might be to have a cash position of no more than 20%. When the stock of some company in that investor's portfolio begins to drop, the agent associated with that stock will notify the manager about these price changes. The strategy could then evaluate whether these changes would break the 20% constraint and take appropriate action viz. recommend buying additional securities.

3.4. SPLENDORS Knowledgebase

Agents and strategies can be extremely specific to the portfolio if desired. However a large number of agents and strategies are quite generic, with just a few parameters controlling the customization. For example, many portfolios may be interested in sampling prices of some of their securities, the only variant being the time interval used to sample. Another example is the dynamic thresholds capability mentioned previously; it takes six parameters: initial high and low value thresholds, initial sampling and notification time intervals, and threshold and interval adjustment increments.

This largely generic behavior of many agents makes it possible to create a collection of such agents which then serves as a knowledgebase library. The same holds true for many strategies. The collection of agents and strategies that are reusable among many portfolios and securities makes it possible for a non-programmer financial expert to create his/her own portfolio managers. SPLENDORS provides a friendly X windows-based user interface to simplify the parameterization of selected components.

For example, a broker may decide to buy a stock for a certain portfolio. SPLENDORS carries out the requisite bindings to make the prices facet for that stock accessible within that portfolio. Then, the user interface allows the broker to choose an agent, >from a library of generic agents, to be associated with that security for that portfolio. It guides the broker through the parameters expected by that agent, thereby providing it the requisite expertise to operate as desired by the broker. Finally, SPLENDORS notifies the strategies associated with that portfolio of the purchase.

4. Using the SPLENDORS Knowledgebase

In the full paper, we will focus on the SPLENDORS user interface and library facilities. We will give a detailed example illustrating the capabilities of the SPLENDORS application system.

5. Conclusion

SPLENDORS provides a unique approach to real time portfolio management. It provides substantial flexibility and extensibility by allowing programmers to write their own agents and strategies, thereby adding to the knowledgebase. At the same time, it provides a way to let the non-programmer build his/her own portfolio managers, which can include security-specific real time monitoring characteristics and portfolio-specific strategy specifications.

The use of Profit as the underlying language provides the programmer enormous convenience through the provision of many powerful constructs useful for real time programming without concern for the details of real time management (handled by the Profit kernel). The actual coding of algorithms and data structures is done in the familiar C language, and existing code can be incorporated.

6. Future Directions

Profit and SPLENDORS are relatively new projects and there is a substantial amount of work that remains to be done. The Profit and SPLENDORS implementation is currently limited to a single process, although multiple objects are supported. We currently use a simulated real time feed as opposed to an actual one. The sample stockbase presently consists of only 50 securities.

Profit needs to be extended to handle multiple processes across multiple machines. SPLENDORS needs to be correspondingly extended to run in a distributed environment (where each workstation on the network belongs to a broker and manages the portfolios of all his/her clients). The notion of libraries needs to be extended to handle generic objects as well as facets. A formal semantics of the Profit language needs to be developed. The current Profit implementation supports a simplified form of dynamic reconfiguration, to add new securities and portfolio managers and change the composition of portfolios while the system is running. A more elaborate method has been designed [2], but has not been implemented.

7. Acknowledgments

Mike Mayer implemented Profit's basic library facility and Vanessa Cole is responsible for the SPLENDORS user interface. George Beltz provided information on the prices feed. We would like to thank Dan Schutzer and Catherine Lassez for useful discussions and Shyhtsun Felix Wu for bringing the dynamic thresholds example to our attention.

Kaiser is supported by National Science Foundation grants CCR-9000930, CDA-8920080 and CCR-8858029, by grants from AT&T, BNR, DEC, IBM, SRA and Xerox, by the New York State Center for Advanced Technology on Computer and Information Systems and by the NSF Engineering Research Center for Telecommunications Research.

8. References

[1] Paola Ciancarini. Coordination Languages for Open System Design. In *International Conference on Computer Languages,* pages 252-260. New Orleans LA, March, 1990.

[2] Brent Hailpern and Gail E. Kaiser. Dynamic Reconfiguration in an Object-Based Programming Language with Distributed Shared Data. In *11th International Conference on Distributed Computing Systems*. Arlington TX, May, 1991. In press. Available as IBM Research Report RC 16232 or Columbia University Department of Computer Science CUCS-048-90.

[3] Gail E. Kaiser and Brent Hailpern. An Object-Based Programming Model for Shared Data. In *1990 International Conference on Computer Languages*, pages 136-144. March, 1990.

[4] Gail E. Kaiser and Brent Hailpern. An Object-Based Programming Model for Shared Data. *ACM Transactions on Programming Languages and Systems* , 1991. In press. Available as IBM Research Report RC 16442 or Columbia University Department of Computer Science CUCS-046-90.

[5] Richard Washington and Barbara Hayes-Roth. Input Data Management in Real-Time AI Systems. In *11th International Joint Conference on Artificial Intelligence*, pages 250-255. August, 1989.