Using OCP to Build Flexible Coordination Protocols: A Transactional Multimedia Application

Tong Zhou Calton Pu Ling Liu
Department of Computer Science and Engineering
Oregon Graduate Institute of Science and Technology
Portland, OR 97291-1000
E-mail: {tzhou,calton,lingliu}@cse.ogi.edu

Abstract

Open Coordination Protocol (OCP) supports the construction of flexible coordination protocols for advanced distributed applications. The design and implementation of OCP combine the ideas of microprotocols, open implementation, and specialization. Using OCP, a large number of efficient and customized coordination protocols can be synthesized. In this paper, we summarize the application of OCP to build custom coordination protocols in extended transaction processing environments. We use OCP in a News Composition example application to illustrate its power and ease of use.

1 Introduction

Recent years have witnessed a variety of advanced applications, in areas like design, automation, and digital media processing. Complex data types, growing sophistication, and flexible interaction within these applications have pushed the limits of previous demands on transactional support. For example, due to the increased necessity of accessing and sharing data over distributed information sources, special system support is necessary for the diverse application requirements on distributed coordination. Consider a multimedia news editing application: one of its tasks deals with live feed, which could be considered as immutable data; while another task deals with multiple video/audio captures from multiple video cameras, requiring multiple iterations of composition script modification and presentation tuning. The distinct characteristics of these two types of data naturally call for different distributed coordination protocols.

We believe advanced transaction models and transactional activity models hold promise for many of today's advanced applications. In analogy to the classic transaction model applied in mission critical applications like banking and airline reservation, these advanced models have well-defined correctness properties that guarantee software reliability. In order to make these advanced models more flexible and practical to meet the diverse requirements of today's advanced applications, we have developed the Reflective Transaction Framework (RTF) [1], Open Coordination Protocol (OCP) [17], Modular Architecture for Recovery Systems (MARS) [3], and Transactional Activity composition Framework (TAM) [8], all of which are built on top of existing commercial transaction or workflow processing products (e.g., Transarc's Encina, Tactica's Caprera).

Open Coordination Protocol (OCP) [17] is a flexible coordination facility for systematically building optimized coordination protocols for distributed advanced applications. We developed OCP in order to support the diverse requirements for distributed coordination in advanced applications. It is an important building block of our concerted effort to implement advanced transaction models on production systems [12].

OCP's "openness" stands for its general functionalities: (1) it can be used to build different coordination protocols for different extended transaction management primitives, (2) it can be used to ensure different global correctness criteria, and (3) it enables different protocol and optimization combinations. Protocol instances built using OCP can execute efficiently, because the built-in quasi-invariants (Section 2.3) guide the specialization process during protocol instantiation time to achieve optimal performance.

In this paper, we summarize our results on OCP-related research, and apply OCP to an advanced multimedia application to illustrate how OCP is used to build and compose different transaction coordination protocols.

---

2 Open Coordination Protocol (OCP)

In designing OCP [17], we adopt a systematic methodology. First, we decompose many existing coordination protocols (e.g., two-phase commit protocol and its variants) into fine-grain *microprotocols* [11]. Second, we follow the *open implementation* principles [6] in designing interfaces for these microprotocols. Third, we use *specialization* technique, being developed in the operating system and programming language communities [14], to compose these microprotocols into OCP and to improve the performance of OCP's instances. In the next few subsections, we summarize the above design methodology while we examine OCP's basic components. Details can be found in [17].

2.1 Microprotocol

Microprotocols are the basic building blocks in OCP. We derive OCP's microprotocols by carefully decomposing many existing coordination protocols. For example, by decomposing the two-phase commit protocol [9, 10], we obtain microprotocols like "Voting", "Deciding", etc. By decomposing the open commit protocol [15], we obtain the "Coordinator migration" microprotocol.

A microprotocol represents a single functional step in a distributed coordination protocol. For instance, the "Voting" microprotocol mentioned above combines the votes from a protocol participant's children to derive a vote representing that participant's own decision about its intended protocol outcome.

By constructing coordination protocols from microprotocols, we obtain several benefits: (1) adaptability and flexibility, e.g., two-phase and three-phase commit protocols; (2) modularity in protocol construction, which in turn brings easy code maintenance and reuse; and (3) enabling systematic and controlled optimizations (e.g., specialization) at the granularity of microprotocols, compared with *ad hoc* optimizations in most existing compilers.

2.2 Functional interface versus Meta interface

For each microprotocol, we distinguish between two types of interfaces: *functional interface* for application programmers, and *meta interface* for system programmers. According to the Open Implementation design principle [6], a functional interface contains a microprotocol's basic functionality, while a meta interface enables a microprotocol's behavior to be modified.

We have carefully designed meta interfaces for OCP's microprotocols to provide flexible customizations. As one example, we associate an upcall with the "Voting" microprotocol to allow customized vote-combining rules to be introduced to alter the behavior of OCP. As another example, we associate an upcall with the "Deciding" microprotocol to allow customized outcome decision rules to be introduced. By adopting meta interfaces in microprotocol design, we have made OCP adaptable to a wide variety of application needs. Consequently, OCP provides coordination protocols for a wide range of distributed ETMs and workflow applications.

2.3 Quasi-invariants and Specialization

OCP can be utilized to implement a wide range of distributed coordination protocols and optimizations. Due to OCP's generality, naive implementations of OCP would carry high overhead. Consequently, an important question is how to make OCP-built protocols efficient. We use *specialization* shown to be effective in optimizing system code [14] as a solution.

Specialization improves software performance by creating and using *specialized versions* of the software that take advantage of invariants and quasi-invariants. An *invariant* is a predicate about the environment that holds true throughout the execution of a module. A *quasi-invariant* is a predicate about the environment that holds true for a period of time and might be falsified at other times. Usually, a specialized software version is generated for a given set of invariants and quasi-invariants, and used when all of its associated invariants and quasi-invariants are true.

We have developed a set of invariants and quasi-invariants that allow us to implement both existing and new optimizations on OCP-built coordination protocols. For example, the NO_OUTCOME_NOTIFY quasi-invariant states that a protocol participant P will not send outcome message to a remote participant Q to ask it to perform corresponding outcome actions. In case this quasi-invariant holds true, the microprotocol "Outcome Notification" is bypassed. This quasi-invariant is used to capture existing optimizations like read-only, presumed-abort (for `abort` outcome), etc.

3 OCP's Applications

In this section, we summarize the application of OCP in building custom coordination protocols for advanced database applications. We use OCP in a multimedia transaction application to illustrate OCP's power and ease of use.

We start with the description of the multimedia transaction application in subsection 3.1. Then in the next two subsections, we show how OCP is used to build the component coordination protocols and optimization relevant to this application. Specifically, we use OCP to build the optimistic commit protocol, presumed-abort commit protocol, read-only optimization, and a distributed `Join` protocol for extended transaction management. Finally, in subsection 3.4, we compose these protocols and optimizations to meet the distributed coordination requirements of the `News Composition` multimedia application.

3.1 A multimedia application

Consider the task of news story composition and broadcast in a digital television studio setting. Typically, the following three steps are involved: 1) identifying relevant components from the original media sources, 2) specifying composition relationships among these components, and 3) mixing and presentation of the data. For the purpose of this paper, we are only interested in step 2). We will refer to it as the `News Composition` process, and we assume the products of this process are scripts that describe the media selections and compositions to be performed for presentation[1]. During this process, the editing persons possibly need to iterate over the steps of source identification, script modification, and trial presentation a few times to finally decide on an optimal presentation specification/script.

With respect to different types of input media sources, the `News Composition` process can be decomposed into several tasks. For example, if we are to produce a news story for a particular athlete right before a match at the Olympics, the source media could include: 1) an introductory news script, in the form of plain text or certain document format, for the anchorperson to read, 2) video of the athlete in several previous matches from multiple video cameras, 3) archival footage of the athlete's training shots, and 4) live feed from the site of the match . In correspondence, the editing process can be decomposed into three tasks: `Text Scripting` that deals with media 1), `A/V Scripting` that deals with media 2) and 3), and `Live Feed Monitoring` that deals with media 4). The complete presentation script produced by the `News Composition` process could start with the anchorperson's introduction of the athlete, flash back to the athlete's training scenes, and then switch to the live broadcast of the match with an on-site reporter's commentary. Furthermore, considering the possible cooperation and collaboration within the studio, a presentation script for certain news story can be well reused or adapted for other news stories, inducing cooperations between the tasks that produce and utilize that script, such as several `A/V Scripting` tasks. In such case, advanced transaction models like the Split/Join model [13] will apply. For example, a task `A/V Scripting'` that previously happened to be editing the presentation script for the training scene in which the athlete appears can "join" the current `News Composition` process for the athlete's story, and in particular, the `A/V Scripting` task.

3.2 Building existing commit protocols and optimizations

In general, many common coordination protocols can be implemented using OCP. To list a few: the presumed-abort (PA) variant of the two-phase commit protocol [9, 10], open commit protocol [15] that introduced the concept of coordinator migration to commit processing, and optimistic commit protocol [7] that allows sites to commit optimistically with the possibility of compensation. For the `News Composition` example, we use OCP to implement two existing commit protocols: optimistic commit and presumed-abort.

Many optimizations on common coordination protocols can also be generated by OCP. For example, many practical optimizations on two-phase commit protocol as described in [16], like read-only, last-agent, ok-to-leave-out, voting-reliable.

3.3 Building coordination protocols for extended transactions

In this subsection, we use OCP to build a new distributed `Join` protocol with read-only optimization, for use in the `News Composition` application. More of OCP's applications for ETMs and transactional workflows can be found in [17, 18]. By applying OCP to construct distributed coordination protocols for extended transactions, we facilitate the deployment of advanced extended models among the wealth of distributed advanced applications.

The split/join transaction model [13] introduces an extended transaction management primitive `Join` for dynamic transaction restructuring. Specifically, when executed in transaction T_1 with transaction T_2 as argument, `Join` delegates all the uncommitted operations of T_1 to T_2. It then deallocates T_1's entries from the system's `transaction table` at all involved sites. "Delegate" means that T_2 takes over from T_1 the responsibility of either committing or aborting those particular operations at its own decision.

We use OCP to develop a coordination protocol for ensuring the correctness of a distributed `Join` execution that involves distributed data (e.g., when the `Join` operation is invoked within a distributed transaction that touches data at two different sites). The protocol operates in two phases. In the first phase (`Preparation` phase), the initiator of the protocol assumes the role of protocol coordinator[2],

---

[1] A script could be eventually replaced by copying all the data referenced to a contiguous space for optimal presentation effect.

[2] This protocol is only for illustration purpose, so we didn't add coordinator migration feature to it.

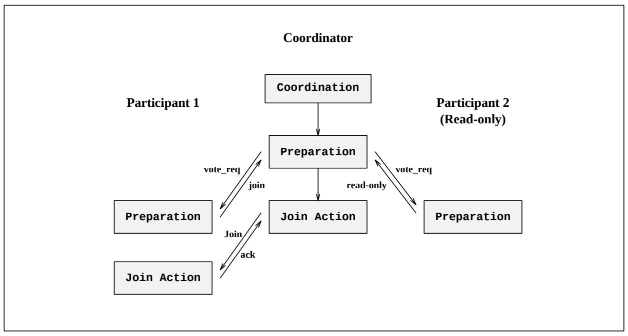

Figure 1: Distributed Join's execution flow

asks other protocol participants to vote regarding the final outcome of the distribute `Join`, and collects their votes. In the second phase (`Join Action` phase), the coordinator looks at the votes to decide each participant's outcome, notifies each participant of its decision, and waits to hear the acknowledgement from each participant, if necessary. There are two types of votes: `join` and `no_join`, and there are two corresponding types of outcomes: `Join` and `No_join`. Each participant acts according to the coordinator in both phases, either voting or carrying out the notified outcome.

Also for illustration, we add a *read-only* optimization to above protocol. For a protocol participant in the distributed `Join`, if none of its operations that are relevant to the `Join` are updates, it is a read-only participant and does not participate in the second phase of the coordination protocol. In addition to `join` and `no_join`, we add a third vote type: `read-only`. This optimization is similar to the read-only optimization for two-phase commit protocols [9, 10], with the difference in the operation set that is used to determine whether a protocol participant is read-only.

Figure 1 illustrates the flow of the new distributed `Join` protocol. The coordinator sends voting requests to both participant 1 and 2. Participant 1 votes `join`, while participant 2 votes `read-only`. The coordinator collects these two votes and discovers that participant 2 doesn't need to show up in the second phase. The coordinator then decides that participant 1 can `Join` and notifies it of this decision. Participant 1 receives the decision, carries out the `Join` action, and acknowledges the coordinator.

3.4 Composing coordination protocols and optimizations

The design of OCP not only allows us to build many coordination protocols and optimizations individually, but also facilitates the flexible composition and interaction among them. The latter is especially appealing to today's advanced transaction applications because of their diverse coordination requirements. In this subsection, we illustrate how flexible protocol compositions can be achieved by way of OCP, in light of the `News Composition` multimedia application in subsection 3.1.

We use a special protocol composition (or composite protocol) to match the `News Composition` application's requirements. The task `Text Scripting` handles text or document editing, with relatively short duration. We thus use OCP to build an optimistic commit protocol [7] for this task to coordinate with the process. Such protocol encourages this task to early commit its updates to the database (in this case, the introduction script), without waiting for other tasks like `A/V Scripting` to commit. In contrast, the task `A/V Scripting` entails composing and lipsyncing relevant audio/video clips, which are large in size and intricate in terms of presentation qualities. Thus, it likely requires more iterations of editing that involves update cancellation and redo. For this reason, we use OCP to build the presumed-abort (PrA) commit protocol [9, 10] for it, in order to save the need of writing many abort log records and sending related messages. The third task `Live Feed Monitoring` generally does not involve updates to either the digital data or the presentation script for its portion, and can be considered as only containing "read-only" operations. Therefore, an OCP-built read-only optimization applies here. Finally, for the cooperation among tasks (e.g., two `A/V Scripting` tasks) in the form of operation `Join`, we apply the distributed `Join` protocol built in last subsection.

Figure 2 sketches the above idea: the `T:News Composition` process is initially composed of several tasks: `A:Text Scripting`, `C:A/V Scripting2`, and `D:Live Feed Monitoring`. The `Transaction Manager` governing each of A, C, and D coordinates with T's `Transaction Manager` using an: optimistic commit protocol, presumed-abort commit protocol, and basic two-phase commit protocol with read-only optimization, respectively. Then, when another task `B:A/V Scripting1` joins task C, B and C's `Transaction Managers` talk a two-phase distributed `Join` protocol.

Note that by using OCP, the same system could use different protocol composition schemes for different applications, rather than using several systems with fixed built-in coordination protocols. We would also like to point out that within each microprotocol, there is code segments that guard the composition constraints in order to prohibit arbitrary protocol composition. However, such guarding policy is "open" via microprotocols' meta-interface so that customization is enabled.

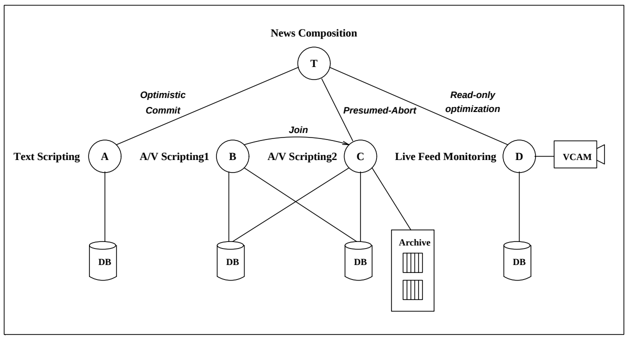

Figure 2: OCP-based protocol composition in the `News Composition` process

4 Summary

In this paper, we have presented the applications of the Open Coordination Protocol (OCP) [17], a flexible coordination facility for advanced distributed applications, in building distributed coordination protocols. OCP contains a set of microprotocols and quasi-invariants that can be modularly composed and adapted to implement a number of existing and new distributed coordination protocols, as well as their compositions and interactions. For illustration, we applied OCP to a multimedia transactional application that deals with news story composition. Specialization technique is employed to make OCP-built protocol instances execute efficiently.

Advanced transaction models hold promise for many advanced applications such as collaborative activities and digital media processing. With OCP being an important component of our effort to implement advanced transaction and activity models on production systems, we hope that it will facilitate the systematic and adaptable deployment of these models for practical distributed applications. We are actively implementing OCP and several instance coordination protocols like distributed `Split` and `Join_tran_group` on Transarc's Encina, integrating with the Reflective Transaction Framework (RTF) [1] for distributed extended transaction management.

References

[1] R. S. Barga and C. Pu. A practical and modular method to implement extended transaction models. In *Proceedings of the 21st International Conference on Very Large Data Bases*, Zurich, Switzerland, September 1995.

[2] P. A. Bernstein. Transaction processing monitors. *Communications of the ACM*, 33(11):75–86, 1990.

[3] S.-W. Chen. *Recovery for Extended Transaction Models.* PhD thesis, Department of Computer Science, Columbia University, February 1997.

[4] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1993.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[6] G. Kiczales. Towards a new model of abstraction in software engineering. In *Proceedings of the IMSA'92 Workshop on Reflection and Meta-level Architectures*, 1992.

[7] E. Levy, H. F. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transaction management. In *Proceedings of 1991 ACM SIGMOD*, pages 88–97, Denver, Colorado, May 1991.

[8] L. Liu, C. Pu, and R. Meersman. A Transactional Activity Model for Organizing Open-ended Cooperative Activities. TR96-21, Department of Computer Science, University of Alberta, Canada, April 1996.

[9] C. Mohan and B. Lindsay. Efficient commit protocols for the tree of processes model of distributed transactions. In *Proceedings of 2nd ACM SIGACT/SIGOPS Symposium on PODC*, Montreal, Canada, August 1983.

[10] C. Mohan, B. Lindsay, and R. Obermark. Transaction management in the R* distributed database management system. *ACM Transactions on Database Systems*, 11(4):378–396, 1986.

[11] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[12] C. Pu, R. Barga, T. Zhou, and S.-W. Chen. Implementing Extended Transaction Models. In *High Performance Transaction Systems (HPTS) Workshop 1997*, Pacific Grove, California, September 1997.

[13] C. Pu, G. E. Kaiser, and N. Hutchinson. Split-transactions for open-ended activities. In *Proceedings of the 14th International Conference on Very Large Data Bases*, 26-37, August 1988.

[14] C. Pu, T. Autrey, A. Black et al. Optimistic incremental specialization: Streamlining a commercial operating system. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, Colorado, December 1995.

[15] K. Rothermel and S. Pappe. Open commit protocols for the tree of processes model. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 236–244, 1990.

[16] G. Samaras, K. Britton, A. Citron, and C. Mohan. Two-phase commit optimizations in a commercial distributed environment. *Distributed and Parallel Databases*, 3(4):325–360, 1995.

[17] T. Zhou, C. Pu, and L. Liu. Adaptable, Efficient, and Modular Coordination of Distributed Extended Transactions. In *Proceedings of the 4th International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, December 1996.

[18] T. Zhou, C. Pu, and L. Liu. A Practical Method to Implement the Restructuring of Transactional Activities. Submitted for publication, 1997.