

RESEARCH NOTE

Cova: a programming language for cooperative applications

YANG Guangxin (杨光信) & SHI Meilin (史美林)

Department of Computer Science, Tsinghua University, Beijing 100084, China

Correspondence should be addressed to Yang Guangxin (email: gxyang@lucent.com)

Received June 28, 2000

Abstract This paper discusses Cova, a novel programming language designed to develop integrative cooperative applications and the issues related to the design and implementation of its runtime system.

Keywords: computer supported cooperative work, meta-groupware, Cova.

With the expense of the scopes and the increase of complexities of CSCW systems, it has been widely accepted that a software platform aiming at easing the development of groupware systems would be critical to both CSCW research and application^[1]. The recognition leads to lots of work on this topic, such as COAST^[2], COCA^[3], Live^[4], XTV^[5], NSTP^[6], COLA^[7], DCW-PL^[8], GroupKit^[9], Renzendous^[10], and so on, to name just a few. All these systems can be termed meta-groupware^[1], which has become a hotspot in the field. We define meta-groupware as a software platform providing a set of general-purpose cooperative computing services based on a specific mechanism for describing the semantics of cooperations and the programming facilities needed for accessing these services^[11].

At the same time, people have also recognized that real-world cooperation is neither simply synchronous nor asynchronous. They are often the integration of cooperation in different modes. This type of cooperation is called integrative cooperation. To the best of our knowledge, there are only a few systems capable of supporting cooperation of this type^[12], nevertheless few meta-groupware systems have this capability. The situation causes great difficulties in developing integrative cooperative applications (ICAs) and hampers the success of groupware systems in practical use.

This paper describes Cova, a programming language for the easy development of integrative cooperative applications. The word Cova comes from the combination of the four italic bold letters. The primary contribution of this work is that it is a first-step toward the generalization of CSCW theories and technologies. This paper details its language features and the runtime system. The coming section explains our approaches to the design of Cova. The second section details its language features. The third section discusses the Cova runtime system, which makes Cova a meta-groupware system. Conclusions and considerations for future work will be given in the last section.

1 Cova approach

Cova's approach to the foregoing challenge originates from the characters of how objects

evolve in cooperative systems. These characters include the following five aspects^[1]. Firstly, objects may play different roles in a cooperative system. Roles can also be changed under certain circumstances. Secondly, objects may flow to different destinations. The flow processes are controlled by a set of rules, either predefined or dynamically formed. Thirdly, objects may be accessed by the destinations they arrive. Tasks of a cooperation are thus be finished via these accesses. Forthly, objects will maintain their correctness specific to application semantics while being accessed. Finally, accesses to objects are not independent, but correlated.

According to these characters, our basic approach is to provide groupware developers with two sets of facilities. One of them is a pure object-oriented programming language for describing the structural and operational semantics of the control and data objects in a cooperative application. The other is a runtime system that provides the cooperative computing services needed at different stages based on the semantics of these objects. The main objective of our approach is to hide from developers the difficulties with implementing cooperative control mechanisms based directly on the services provided by traditional distributed software platforms.

2 The Cova language

According to the definition given above, a meta-groupware system should provide the mechanisms needed for describing the semantics of cooperation. The Cova language is such a mechanism. It is designed as a programmable method for describing the Cova model, which is a novel mathematical system for modeling real world cooperations. In this section, we will first discuss the Cova model, then the Cova programming language.

2.1 Cova cooperation model

As have been stated, a cooperation model is a formal system for describing cooperation in real world. They can be used to describe the properties of different stages of a cooperation as well as the control and data flows among them. Services provided by a meta-groupware system are completely based on its cooperation model. It is of critical importance to the functionality and scope of a meta-groupware system. A perfect model should have the capability and flexibility to model uniformly cooperation in different modes and dynamic work settings. With the Cova model, a cooperation process is modeled as a set of interrelated Cova activities. A Cova activity is an autonomous computing entity with its objective, its lifecycle, its rules regarding how its internal state changes, and its rules describing how it interacts with the environment where it resides. All these four parts together describe the static and the dynamic properties of a stage in a cooperation process.

The objective is the desired result of an activity. It can be a document, a drawing, or some other artifacts. The artifact is part of the desired results of the whole cooperation process. The Cova object description language can be used to describe and implement the structural and operational semantics of these artifacts. Based on the semantic information, flexible control over the cooperation can be achieved when objectives are accessed by activity participants.

The state rules specify how the state of an activity can be changed. These rules include the activation condition, i. e. under what condition the activity can be activated; the specification of participants, i. e. who (a user or a group of users) or which applications can participate in accessing the objective maintained by the activity; the role definitions, i. e. in which way (read, write, or execute) the participants can access the objective.

The interaction rules specify how an activity interacts with the environment where it resides. The environment of an activity is the set of all other activities in the same cooperation. Interactions include information exchange and state control. An activity can get the required information from the environment. On the other hand, it can also provide information for other activities. Information exchange may lead to the state change in activities. Autonomous activities are thus interrelated by the interactions among them, which drives the evolution of the whole cooperation.

The lifecycle refers to the different stages through which the activity navigates. A Cova activity is a historically evolving computing entity. Once an activity is created, it may be activated, disabled, completed, terminated, or aborted. The activity state determines whether the activity object can be accessed. An activity object can be accessed only when the activity is activated. Otherwise, it is invisible to the participants. The evolution of activity state may be driven internally by its objective, or externally by some specific operations.

According to the description above, we can see that a Cova activity actually defines the rules regarding how the data object (in this case, the objective) can be accessed by the destination to which it flows. It also provides a framework for the destination to finish this access. A Cova process definition is a Cova model-based formal representation of a cooperation process. It will act as the control object in the Cova system. The Cova programming language is then designed to describe these objects. In the following section, we will use a simplified example to show how the language can be used.

2.2 Cova programming language

As a programmable method for describing the Cova model, the Cova language includes two independent yet tightly related parts. One of them is the Cova object description language (CODL), which can be used to describe the structural and operational semantics of activity objects. The other one is the Cova cooperation description language (CCDL), which can be used to describe how a process can be divided into different stages, the control policies of each stage, and the relationship among them. Due to page limitations, here we only give a brief overview of the language. A more detailed description can be found in ref. [11].

CODL is purely object-oriented. It implements an extended version of the object model defined by ODMG^[13]. It supports multiple inheritance, polymorphism, encapsulation, and local class definition. The structural and operational semantics of an atomic object are described with its attributes and operations. An attribute has a type, which can be a primitive type, i. e. Boolean, byte, char, tiny, short, int, long, float, double, string, binary, or a collection object type, i. e. set, bag, dictionary, list, array, tree, and graph, or an atomic object type. An operation can be implemented with Cova's object query and manipulation language. Besides the conventional commands, e. g. variable declaration, flow control, exceptional handling, there are four additional ones, foreach, insert, delete, and update, which can be mixed with other commands to navigate and manipulate collection objects.

CCDL is declarative, object-oriented, and CODL-based. Its basic construction unit is a process definition, which may include class definitions, activity definitions, and local process definitions. Class definitions are given in CODL. They define the structural and operational semantics of the objects used at various stages of the process. A local process definition is a mechanism that can be used to combine several related activities into a meaningful logical unit, which can then be specified as a sub-process of current process definition. An activity definition in-

cludes the specification of the class to which the objective belongs, the activation condition, the role definitions, the participant definitions, and the trigger definitions. The triggers define how an activity interacts with other activities based on invocations to the operations defined for activity objects. It is in this sense that we say CCDL is based on CODL.

CCDL and CODL are the direct results of Cova's attempt to make a clear separation between the coordination part and the computation part of a cooperation process. However, the tight relationship between them makes it possible for Cova to provide the flexibility which otherwise is unavailable in purely coordination-oriented languages, such as COCA^[3] and DCWPL^[18].

Fig. 1 illustrates how the language can be used to describe a review process. For simplicity, details of the two class definitions are omitted. Two activities are presented. The synchronous group defined in the first activity states that all users in the system can participate in composing the proposal in real-time mode. The first trigger states that when the proposal is finished, it will be passed to the second activity, namely AReview. After a project reviewer defined in the second activity finishes his/her review on the proposal, the result will be passed to the first activity by an invocation to CProposal::SetReviewResult(...). All participants of the first activity will then receive an e-mail message containing the review result.

```

public process PProjectReview startsat AApplication
{
    public class CProposal {...};
    public class CReport {...};
    activity AApplication handles CProposal
    {
        synchronous group Applicants as all;
        trigger AReview. SubmitProposal(this) when after complete where true;
        trigger system. EMail. SendMail ( activity. GetUsersList(), this. GetReviewResult ()
            when after execute SetReviewResult where true;
        ...
    };
    activity AReview handles CReport startswhen (this. GetProposal ()! = null)
    {
        role ProjectReviewer as {grant all on all};
        trigger AApplication. SetReviewResult (this. GetReviewResult( ))
            when after complete where true;
        ...
    };
    ...
};

```

Fig. 1. An application process described in Cova.

Cova does not provide the mechanisms for specifying which applications should be used to access activity objectives. Developers have full freedom in choosing these applications. For example, the proposal can be composed of a synchronous co-authoring tool. In this case, the definition of class CProposal can be greatly simplified. Only one attribute of type binary or string will do. However, in this case, Cova will not be able to understand the semantics of the proposal object. Therefore, it will be quite difficult to design flexible control policies based on the activity object. If these policies are required, CProposal can be enhanced by adding more attributes and operations.

Process definitions are inheritable. Therefore, it is possible for developers to model cooperation from a general level to a more specific level. Other object-oriented features like encapsulation

and polymorphism are also implemented in CCDL^[11]. Existing activities can create dynamically new activities, which will become part of the process definition as if they were predefined. The declarative nature of CCDL makes it possible for the control policies to be modified dynamically. These dynamic features make Cova capable of accommodating to ever-varying working conditions.

3 Cova runtime system

Cova by itself is only a formal method for describing the semantics of cooperation processes. These semantics are interpreted by the Cova runtime system, which provides the cooperative computing services needed for enacting process instances. This section discusses the services provided by the runtime system as well as how it works.

3.1 Cova service sets

First, it is necessary to identify the minimum set of services that makes Cova a meta-groupware system capable of supporting ICAs. Then based on these core services, a number of extended services are introduced to make the platform more powerful and practical. This subsection discusses the semantics of these services. Fig. 2 depicts the architecture of these services.

3.1.1 Core service set. The core set provides the basic services needed for building ICAs. Based on the cooperation model, the core set includes three types of services.

Activity control. As have been stated, a Cova activity is a computing entity with a lifecycle. Activity control manages the creation, activation, deactivation, and termination of process instances and activities as well as the message exchanging and the synchronization among them. The relationship between process instances and activity control is similar to that between processes and the process management subsystem in operating systems. The difference is that the enactment of activities is driven by events generated at different times. Therefore, the mechanisms for activity control are quite different from the ones for process management.

Replication control. Activity objects can be shared by multiple cooperators in a synchronous session. For short response time and increased reliability, activity objects are replicated at each participant's site. Replication control manages the sessions and provides reliable multicast delivery so that the logical equivalence among the replicated objects is preserved when an activity object is dynamically opened, closed, or saved. User operations on a replicated object will be translated into invocations to operations of its class definition. The invocations will then be multicast to other sites in the same session for execution so that the awareness among cooperators can be achieved.

Concurrency control. The primary goal of this service is to maintain the consistencies while an activity object is being shared in a real-time session with fully-replicated architecture. In this case, there is only one thread accessing the replicated copy at each site. User operations will be multicast to other sites after local executions. During the executions of these operations, three types of inconsistency may arise, i. e. causal inconsistency, inconsistency among the results produced by an operation at different sites, and inconsistency among the final states of replicated objects. All these inconsistencies will be eliminated by oodOPT, a Cova's semantic-based solution

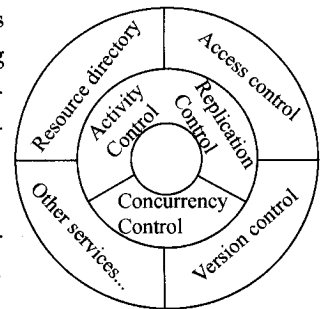


Fig. 2. The Cova core and extended service sets.

to concurrency control in fully-replicated architecture^[14].

3.1.2. Extended service set To make Cova a more powerful platform for building practical cooperative applications, a number of extended services are implemented in addition to the core service set. These services are described hereinafter.

Resource directory. External resources will be used during the enactment of cooperation instances. Cova maintains a resource pool consisting of a set of predefined resource objects. A Cova resource object represents an information system or a real world object, e.g. a user, a group, or an organization. It provides the information about a resource and the interfaces for accessing it. A resource object is implemented with a standard CODL class. Therefore, new resources can be introduced as needed. This makes Cova an extensible and open platform.

Access control. The first step for access control is the authentication when users are connecting to the Cova system. However, the role a user plays may vary from time to time, for s/he may have different authorizations in different activities. In the Cova programming language, access control policies are defined at the activity level. This provides the maximum flexibility for application developers. At the same time, a set of meta-data, i.e. the context properties of activities, objects, and process instances, are available for constructing control policies specific to application semantics.

Version control. Cova implements a storage subsystem to record the persistent states of cooperation instances. Version control maintains multiple versions of an activity object so that the evolution history of a cooperation process could be reflected. Another issue in version control is to maintain the correlation, or more specifically the dependency, among multiple versions of activities objects so that we can get a more comprehensive view of process instances.

We believe that the services outlined above are only necessary for building ICAs. It is not to say that they are sufficient or complete. New services may be introduced as Cova's application area expands. In the next section, we will depict how the whole system works with a runtime scenario.

3.2 Cova virtual machine

The Cova virtual machine (CovaVM) is designed to implement the mechanisms needed for enacting cooperation instances through interpreting the semantics of process definitions. It manages these instances as top-level objects and is responsible for their creation, message exchanging, and message execution. In this sense, CovaVM is not simply an instruction interpreter, but a self-organized distributed system.

Fig. 3 depicts with a simple scenario how CovaVM works. The server maintains three dependant spaces: the definition space, the object space, and the process space. The definition space stores all the intermediate codes generated from Cova source codes by the Cova compiler. The object space maintains the persistent states of Cova process instances and activity objects. Activity objects can be accessed only from within the context of an activity. The process space contains all cooperation instances and provides the mechanisms needed for their enactment. In this scenario, there are n instances in the process space, namely p_1, p_2, \dots, p_n respectively. They can be the instances of different processes. A process can have multiple instances. Each activity in an instance maintains an object, which belongs to a Cova class. For example, the object o_i maintained by A_i in instance p_1 belongs to c_i , where i ranges from 1 to 3.

An object can be opened when its owner activity is active. CovaVM supports three modes for

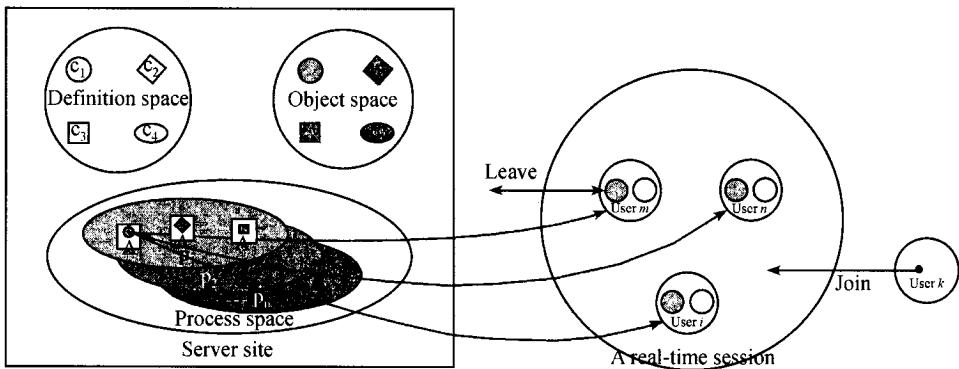


Fig. 3. A runtime scenario of Cova virtual machine.

an open operation: read-only, exclusive and cooperative. A read-only open returns the latest state of an object. Opening an object exclusively prevents other users or applications from accessing it before the lock is released. A cooperative open enables the object to be shared by multiple users or applications in a dynamically organized real-time session. In this case, the object and its class definition are replicated at each cooperative site, as shown in the right part of fig. 3. Based on the structural and operational semantics of the object, CovaVM guarantees the logical equivalence of the replicated copies as well as the logical equivalence and causal dependency of user operations when they executed at multiple sites.

This scenario also outlines the primary approach to make Cova an integrative platform. Activity control acts as the skeleton for enacting cooperation instances, while replication and concurrency control as well as other services together flesh the skeleton out to make the platform more flexible and powerful. To one end, a process can have only one activity. If the activity object is always opened in sharing mode, the application works in synchronous mode. If it is always opened in exclusive mode, the application works in a single-user mode. To the other end, activity objects can be limited to be opened only exclusively. In this case, the application is asynchronous. Fig. 4 shows the relationship among the sharing mode of activity objects, the number of activities in a process, and the cooperation mode in which applications work.

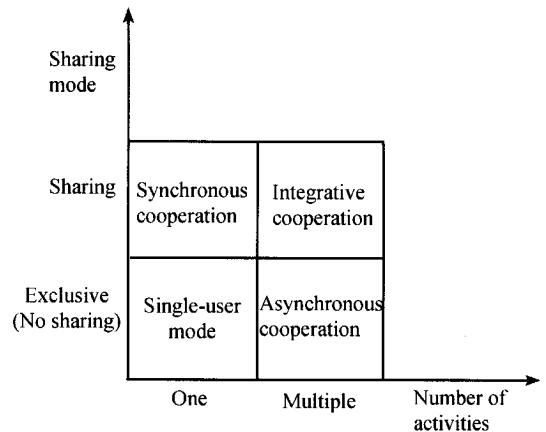


Fig. 4. A taxonomy of cooperation modes supported by Cova.

Ref. [11] discussed in full detail the formal descriptions of CovaVM and the underlying algorithms. Due to page limitations, they will not be repeated here.

4 Conclusion and future work

This paper discusses Cova, a novel cooperative application-oriented programming language, including its underlying cooperation model, the programming language, and the runtime system.

Compared with other related work, the activity-based cooperative computing model embodies a change, i. e. to turn networked computing facilities into an infrastructure that supports the flexible interaction and cooperation among multiple autonomous systems capable of some operations. For the time being, the Cova compiler and the virtual machine, including the Cova interpreter and the algorithms for the core services have been implemented on the Win32 platform. The compiler and interpreter of CODL have also been used in our general-purpose workflow management system^[15]. Future work includes developing an integrative development environment based on CovaVM as well as some practical applications.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant No. 269773029) and National High-Tech Research and Development Plan under contract No. 863-306-ZD-10-2B and 863-306-ZD-02-03-1.

References

1. Ellis, C., Wainer, J., A conceptual model of groupware, in Proc of ACM Conf on Computer Supported Cooperative Work, Chapel Hill, 1994, 79—88.
2. Schuckmann, C., Kirchner L., Schummer J. et al., Designing object-oriented synchronous groupware with COAST, in Proc. of ACM Conf. on Computer Supported Cooperative Work, Cambridge, 1996, 30—38.
3. Li, D., Muntz R., COCA: collaborative objects coordination architecture, in Proc. of ACM Conf. on Computer Supported Cooperative Work, Seattle, 1998, 179—188.
4. Banavar, G., Doddapaneni S., Miller, K. et al., Rapidly building synchronous collaborative applications by direct manipulation, in Proc. of ACM Conf. on Computer Supported Cooperative Work, Seattle, 1998, 139—148.
5. Abdel-Wahab, H. M., Feit, M. A., XTV: A framework for sharing X Window clients in remote synchronous collaboration, in Proc. of IEEE Conference on Communications Software: Communications for Distributed Applications and Systems, Chapel Hill, 1991, 159—167.
6. Patterson, J. F., Day M., Kucan, J., Notification servers for synchronous groupware, in Proc. of ACM Conf. on Computer Supported Cooperative Work, Cambridge, 1996, 122—129.
7. Trevor, J., Rodden, T., Blair, G., COLA: A lightweight platform for CSCW, in Proc. of European Conf. on Computer Supported Cooperative Work, Milan, 1993, 15—30.
8. Cortes, M., Mishra, P., DCWPL: A programming language for describing collaborative work, in Proc. of ACM Conf. on Computer Supported Cooperative Work, Cambridge, 1996, 21—29.
9. Roseman, M., Greenberg, S., Groupkit: A groupware toolkit for building real-time conferencing applications, in Proc. of ACM Conf. on Computer Supported Cooperative Work, Toronto, 1992, 43—50.
10. Hill, R. D., Brinck, T., Rohall, S. L. et al., The Rendezvous architecture and language for constructing multiuser applications, ACM Trans. on Compute Human Interaction, 1994, 1(2): 81.
11. Yang, G. X., Research on meta-groupware—the Cova programming language and system, PhD Dissertation, Beijing: Tsinghua University, 2000.
12. Weber, M., Partsch, G., Hock, S. et al., Integrating synchronous multimedia collaboration into workflow management, in Proc. of ACM SIGGROUP Conf. on Supporting Group Work, Phoenix, 1997, 281—290.
13. Cattell, R. G. G., Barry, D., Bartels, D. et al., The object database standard: ODMG 2.0, San Mateo: Morgan Kaufmann Publishers, 1997.
14. Yang, G. X., Shi, M. L., Semantic-based approach for concurrency control in fully-replicated architecture, Journal of Computer Science and Technology, to appear in 2001.
15. Shi, M. L., Yang, G. X., Xiang, Y. et al., A web-based workflow management system, Chinese Journal of Software, 1999, 10(11): 1148.