CoMex – A Mechanism for Coordination of Task Execution in Group Work

Hilda Tellioğlu

Vienna University of Technology, A-1040 Vienna, Austria

Abstract. CoMex (Coordination Mechanism) is a system to coordinate the execution of tasks accessing coordinable entities. It uses coordination rules to describe the temporal and logical order of tasks performed in a cooperative work setting. These rules coordinate semantic dependencies between work activities carried out by different users. The coordination rules are implemented in a relational database. This makes CoMex unique and easy to integrate into an existing application. By illustrating a case from a real work setting we show how these rules can be created, which methodology can be applied for their production and how CoMex can be used in a web application. We also describe the implementation of CoMex and its architecture.

1 Introduction

Collaboration within a workgroup can only be established when several communication, coordination and cooperation mechanisms are provided by the system in use. No matter whether computerized or paper-based, these mechanisms have a great impact on work processes and must therefore be designed carefully.

Efficiency in the collaboration of workgroup activities requires coordination mechanisms to ensure that dependencies occuring because of contextual, temporal or organizational reasons are considered accordingly. Some researchers tried to identify and classify coordination processes [9]. Others tried to describe coordination problems within complex work processes. To them this is the precondition to provide solutions to these problems. Etcheverry et al. defined a topology of coordination problems and tried to solve them by means of well-known coordination forms [10]. They provided a catalogue of coordination patterns by taking the coordination situations identified by Malone and Crowston [14].

There are also several coordination models and languages which are mostly categorized by distinguishing between data-driven (like Linda [4], Sonia [7], Laura [19], Ariadne [12] etc.) and process-oriented coordination models (like IWIM [1], MANIFOLD [2] [3]) [17, p.35ff]. The focus of these approaches is to overcome the complexity of providing coordination mechanisms on a technical level. Most of them do not consider how the cooperative work is in fact carried out.

Tellioğlu H. (2004).
CoMex – A Mechanism for Coordination of Task Execution in Group Work.
In *Proceedings of the 1st International Workshop on Computer Supported Activity Coordination*, pages 13-20
DOI: 10.5220/0002670200130020

In this paper we will present the coordination mechanism called CoMex and show the methodology we applied to design the coordination setting for a work environment. *CoMex is a mechanism to coordinate the execution of tasks accessing coordinable entities. It uses coordination rules to describe the temporal and logical order of tasks performed in a cooperative work setting by several actors and to coordinate semantic dependencies between work activities.*

We take a real work domain as an example to illustrate our methodological approach. Our case is about an international insurance company that uses a management information system (MIS) with workflow functionality. Some requirements are management of information packet requests and enrollment forms clients send, or management of policies they offer and maintain. The system is a web-based application. Some of the artifacts are inquiry requests, enrollment forms, policies, contacts, dependents, outstanding items etc. Figure 1 illustrates the basic activities carried out after receiving an enrollment form.[1]

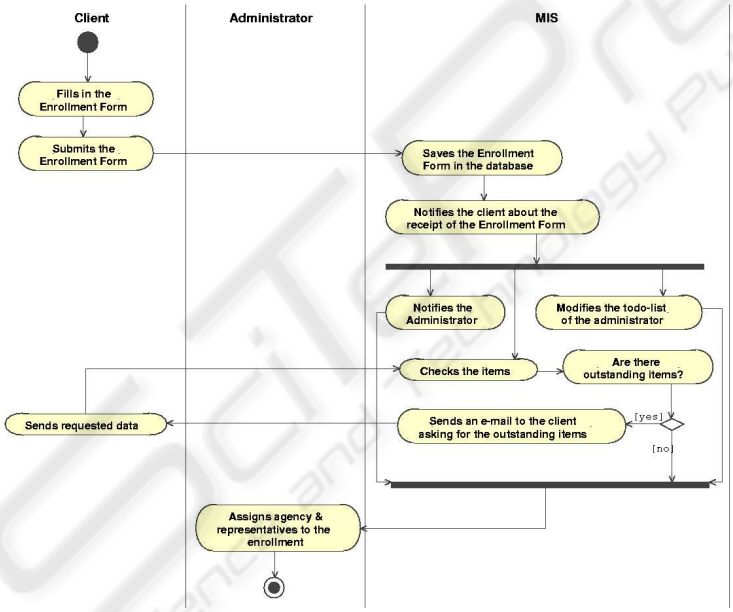

Fig. 1. Activity diagram to illustrate the enrollment processing.

The next section shows how we carried out the systems design in our case. This is the methodology that we found very effective and useful if one has to introduce a coordination mechanism into a work environment. Section 3 presents the coordination mechanism CoMex. In the same section its implementation is presented in detail before concluding the paper.

[1] For simplicity reasons the diagram does not show the whole activity.

2 The Methodology

The conceptualization of a work domain is the first step to understand its objects and relations between its actors. The entities that may exist in that domain and the relationships among those can be named and described by means of a domain conceptualization. This provides a vocabulary for representing and communicating knowledge about that domain [11, p.2]. In [18] the notion of ontology is used to model artifacts, create hierarchies and layers as a conceptual model that is processable by computers. [15] had developed an approach to identify elements to be coordinated (static coordination) and how these entities may be coordinated (dynamic coordination). We have an object-oriented approach to conceptualize the artifacts of the work domain. We create *use case diagrams* with UML to understand and show what users want to do with the system entities, how they carry out their work when they deal with their artifacts, what their requirements are. In our example, users want to receive, modify, send out, submit an enrollment form, receive an inquiry request and send an information packet to the client, be informed when a new enrollment form or inquiry request is received, calculate the effective date of the policy, contact the corresponding persons like clients, agents, representatives etc.

Next, we create a *data dictionary* that describes concepts used in the system and their relations to each other. After identifying classes, their attributes and relations we produce a *domain model* that we may modify several times. Next step is to define the constraints that the system has to fulfill. Relations, associations and hierarchies between entities include restrictions and norms. We consider these in the design process. We use the concept of *rules* that derive from the relations defined in the domain model. Rules express the (hidden) dependencies between entities. They describe what needs to be guaranteed by the system in order to achieve consistency and correctness in the domain model. Rules emerge when users talk about dependencies between their artifacts and activities. Rules need to be refined and communicated between users and systems designers until all are convinced of the necessity, correctness and usefulness of them.

Rules can in a first step be presented in a natural language. This makes their exchange between users and designers easy. For instance, some of the rules for enrollments are[2]:

R_1: If an enrollment form is received, check all items attached to the enrollment.
R_2: If an enrollment form is received, the administrator must be noti¯ed.
R_3: If an enrollment form is received, the todo-list of the administrator must be modi¯ed.

Finally, we create *activity diagrams* to show the tasks to be performed and their temporal and logical order (see Figure 1). This way we design the specific task execution and define the work order.

[2] For simplicity only the rule R 1 is considered in this paper.

3 CoMex – The Coordination Mechanism

The coordination theory applied in CoMex is based on processes that consist of resources, activities and dependencies [14]. A dependency is a relation among activities. There are three types of dependencies: flow, sharing and fit. A flow dependency occurs when one activity's output is a resource that is used by another activity. When multiple activities use the same resource we talk about a sharing dependency. A fit dependency happens when multiple activities together produce a single resource. "Using these three basic types, any process can be decomposed into a structure of activities and dependencies." [13, p.2]. Coordination is "managing dependencies between activities" [14, p.90]. Dependencies can be described by using temporal relations between tasks. Allen defined seven primitive relations between pairs of tasks [5]. These descriptive, mutually exclusive relations can be applied over time intervals T_1 and T_2:

- T_1 equals T_2 when they start and end at the same time.
- T_1 starts T_2 when both start at the same time, T_1 ends before T_2.
- T_1 finishes T_2 when T_1 starts after T_2 and both end at the same time.
- T_1 meets T_2 when T_2 starts when T_1 ends.
- T_1 overlaps T_2 when T_1 starts before T_2 ends and T_1 ends before T_2.
- T_1 is during T_2 when T_1 starts after and ends before T_2.
- T_1 is before T_2 when T_1 ends before T_2 starts.

Allen also used a set of axioms to create a temporal logic based on these relations [6]. Raposo and Fuks extended Allen's seven primitives by introducing active, passive, blocking and resource management interdependencies. They created a model by proposing Petri-Net-based coordination mechanisms to deal with these task interdependencies [16].

The coordination mechanism that is shown in this paper is based on task interdependencies. The passive interpretation of these dependencies [16] are not used because they do not support active handling in the task coordination.

By analyzing the activity diagrams we can define *coordination rules*. These are rules that are derived by analyzing the temporal and logical order of tasks that must be performed in order to implement a use case in an application. In our example we can define the following coordination rules (CR_i):[3]

CR_1: receiveEnrollment meets checkItemsOfEnrollment
CR_2: receiveEnrollment meets notifyAdministrator
CR_3: receiveEnrollment meets modifyTodoListOfAdministrator

CoMex uses a relational database to store the coordination rules. Figure 2 shows the tables and relations in the CoMex database. Coordination rules (table `CR`) are coordination laws [17] that determine how active entities are coordinated using the coordination media. They specify the semantics of the model's framework.

Before the coordination rules can be entered into the database, CoMex needs to know about the entities managed in the system. Entities can be saved in the domain's

[3] A simple notation is used here.

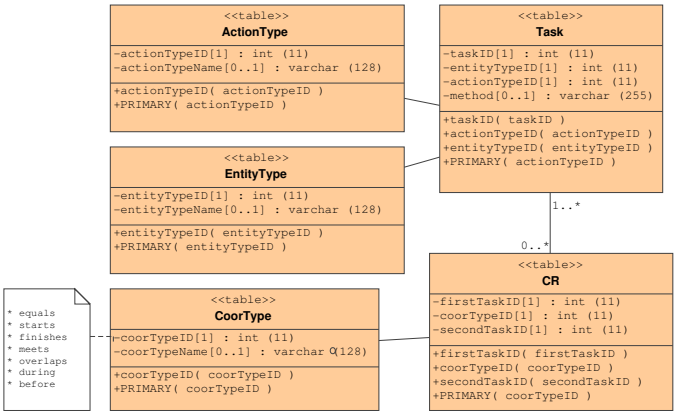

Fig. 2. The database structure of CoMex.

own database, only their types must be entered into the CoMex database. Entities used in this example are `Enrollment`, `Info Packet Requests`, `Policy`, `Outstanding Item`.

To associate entities with rules, tasks are introduced. Certain tasks are assigned to certain entities. Figure 2 shows the relation between entity and action types, tasks and coordination rules. Some action types in our case are `receive`, `submit`, `modify todo-list`, `notify`, `check items`.

To create coordination rules tasks are needed. Tasks include actions to different entity types and the name of the method that must be invoked in the application when the task is carried out. For some tasks no methods are necessary and the value is set to `null`. The `entityTypeID` of `Enrollment` is "1". The `actionTypeID` of `receive` is "11". We need `"insert into Task values (1,1,1,null);"` for `receiveEnrollment` and `"insert into Task values (13,1,11,'checkItemsOfEnrollment');"` for `checkItemsOfEnrollment`.

Coordination rules consist of tasks and a coordination type. Coordination types are `equals`, `starts`, `finishes`, `meets`, `overlaps`, `during`, `before`. In our case we try to create coordination rules for the entity `Enrollment`. If an enrollment is received – which happens by means of a HTML-form in the web application which on the one hand sends an e-mail to the administrator and on the other hand stores the data entered into the database – three tasks must be executed. We show here only the one which starts to check all items that need to be attached if one of the questions in the enrollment form is answered with "Yes" (CR_1). The `coorTypeID` of `meets` is "4" and we need the entry `"insert into CR values (1,4,13);"` in the database.

CoMex is implemented in Java (JDK 1.4). We used MySQL (Version 3.23.55) as database, The Tomcat 4 Servlet/JSP Container (Version 4.1.24), The Apache Jakarta Struts (Version 1.1) as an open source framework for building the web application. Figure 3 shows the architecture of CoMex and its interfaces to other systems. The client tier provides an interface to interact with the application. The interaction includes submitting a request and receiving a response from the middle tier. Web container with the web pages, servlets and beans (computation part) and CoMex are located in the middle tier. CoMex is separated from the computation part, has its own database

(CoMexDB) and its own interfaces enabling configuration (CoMexAdmin) and monitoring (CoMexMonitor). CoMexService is the main part of the mechanism in which coordination rules are checked for a given task. The domain database and the CoMexDB are located in the enterprise information system (EIS) tier.

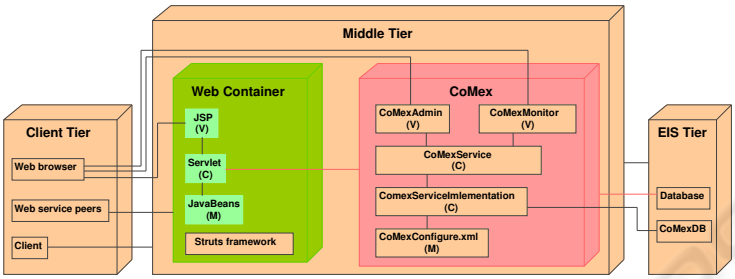

Fig. 3. System architecture. (M = Model, V = View, C = Controller)

If a client fills in the Enrollment Form in the web site and the data entered is validated, CoMex is instantiated by the Action Servlet of the form Enrollment. The method EnrollmentAction.execute() is called. It returns an ActionForward object which is a wrapper around the physical resource specified in the configuration file.[4] EnrollmentAction.execute() sets all attributes of an Enrollment bean getting the validated values from the EnrollmentWebForm given to the method by the parameter ActionForm and stores the current Enrollment in the data store. We implemented a service interface that Action classes use instead of interacting with the persistence framework directly [8].

After successfully storing the Enrollment in the database, the EnrollmentAction.execute() gets a CoMex service and starts the coordination process by invoking the method ComexService.executeCR(). This has three parameters: the object which is the instance of the entity for which coordination rules must be retrieved from the database (in our case the enrollment), the type of this object and the action type.

What happens in the method ComexService.executeCR()? It connects to the CoMex database, selects all coordination rules associated to the entity type given. By each coordination rule (a selected row from the table CR) it looks for any method declared in the corresponding tasks. Some of the tasks do not have any method defined. That means it is not necessary to execute a specific action by CoMex. But if there is a method, its name is retrieved and saved in the string buffer methodStr. Then ComexService.executeCR() searches for the full name of the method by using Class.forName() of the Service Implementation which it reads from its configuration file (CoMexConfigure.xml). The XML file is used to configure classes that are relevant for CoMex.[5] Then it invokes the method by instantiating the appropriate

[4] The ActionForward class represents a logical abstraction of a web resource which can be a Java Server Page or a Java servlet.

[5] For simplicity reasons dealing with the configuration file is not shown here.

class. In our example, to check the items needed for the enrollment form the method `checkItemsOfEnrollment()` is invoked. It looks for documents on the server that had to be uploaded by the client in case of a positive answer to any of the questions in the enrollment form. If the document cannot be found the system sends an e-mail to the client asking for the outstanding items.

4 Conclusions

In this paper we defined coordination rules which describe the temporal and logical order of tasks performed in a cooperative work setting. These are used to coordinate semantic dependencies between work activities carried out by different users. We also showed – by illustrating a case from a real work setting – how these rules can be created, which methodology can be applied and how this mechanism can be implemented technically.

Coordination problems are context-dependent. Etcheverry et al. had a general approach, without considering the context in which coordination takes place [10]. We tried to introduce a methodology that enable user involvement in identifying context-dependent coordination situations. Users can specify dependencies between tasks they carry out. The rules they define can easily be implemented in CoMex. CoMex is a safe, cheap and easy way to control business processes: Dependencies between activities are coordinated and tasks that must follow other tasks or must be executed at the same time are invoked automatically. The system must guarantee that data is consistent, necessary steps are taken and nothing is forgotten or missing. Additionally everything must happen at the right time.

We argue that following issues are important in this context:

- It is not necessary to take care of any type of task dependency. Controlling and coordinating the share of common resources is the responsibility of the database management system used. In fit dependencies we find activities running simultaneously or overlapping which are then synchronized to a given point of time. One can easily map fit dependencies into flow dependencies using the seven relations defined by Allen. The only dependency type that concerns any coordination mechanism is then the flow dependency. That is why we considered in CoMex only the flow dependency.
- Is is important to apply UML in systems design, it enables better communication between users and designers. But it is not necessary to produce too many diagrams when these do not enhance the quality and efficiency of the process.
- It is still a problem to represent rules with UML. For this purpose, a formal language is needed.
- To separate coordination from computation the transfer of domain specific data to the coordination mechanism must be avoided. The interface between CoMex and the application or other systems must dynamically create the necessary information flow in the run time. In the current version of CoMex we still use a configuration file that we want to eliminate in the future.

In this paper we presented a running version of CoMex. `comex.jar` can be used by any object-oriented web application. However, the development of CoMex is not finished yet. `CoMexAdmin` and `CoMexMonitor` are just prototypes and must be refined and completed.

References

1. Arbab, F.: The IWIM Model for Coordination of Concurrent Activities. First International Conference on Coordination Models, Languages and Applications (Coordination'96), 15-17 April, (1996) 34-56
2. Arbab, F., Blom, C. L., Burger, F. J., Everaars, C. T. H.: Reusable Coordinator Modules for Massively Concurrent Applications. Europar'96, (1996) 664-677
3. Arbab, F., Herman, I., Spilling, P.: An overview of Manifold and its implementation. Concurrency: Practice and Experience **5(1)** (1993) 23-70
4. Ahuja, S., Carriero, N., Gelernter, D.: Linda and Friends. IEEE Computer, August (1986) 26-34
5. Allen, J. F.: Towards a General Theory of Action and Time. Artificial Intelligence **23** (1984) 123-154
6. Allen, J. F.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM **26(11)** (1983) 832-843
7. Banville, M.: Sonia: an Adaptation of Linda for Coordination Activities in Organizations. First International Conference on Coordination Models, Languages and Applications (Coordination'96), 15-17 April (1996) 57-74
8. Cavaness, C.: Programming Jakarta Struts. O'Reilly (2002)
9. Crowston, K.: A Taxonomy of Organizational Dependencies and Coordination Mechanisms. http://ccs.mit.edu/CCSWP174.html Technical Report, 174, Massachusetts Institute of Technology, Center for Coordination Science (1994)
10. Etcheverry, P., Lopisteguy, P., Dagorret, P.: Pattern-Based Guidelines for Coordination Engineering, DEXA 2001, LNAI 2113, eds. Mayr H. C. et al., Springer-Verlag Berlin Heidelberg (2001) 155-164
11. Farquhar, A., Fikes, R., Pratt, W., Rice, J.: Collaborative Ontology Construction for Information Integration. KSL 95-63, Technical Report, Knowledge Systems Laboratory, Stanford University (1995)
12. Florijn, G., Besamusca, T., Greefhorst, D.: Ariadne and HOPLa: Flexible Coordination of Collaborative Processes. First International Conference on Coordination Models, Languages and Applications (Coordination'96), 15-17 April (1996) 197-214
13. Hayashi, N., Herman, G.: A Coordination-Theory Approach to Exploring Process Alternatives for Designing Differentiated Products. http://ccs.mit.edu/wpmenu.html Massachusetts Institute of Technology, Sloan School of Management, Center for Coordination Science (2002)
14. Malone, T. W., Crowston, K.: The Interdisciplinary Study of Coordination. ACM Computing Surveys **26(1)** (1994) 87-119
15. Muccini, H., Mancinelli, F.: Eliciting Coordination Policies from Requirements. SAC 2003 (2003)
16. Raposo, A. B., Fuks, H.: Defining Task Interdependencies and Coordination Mechanisms for Collaborative Systems. Coopertive Systems Design. A Challange of the Mobility Age, eds. Blay-Fornarino, M. et al., IOS Press (2002) 88-103
17. Schumacher, M.: Coordination Models and Languages. Objective Coordination in MAS Engineering, Springer-Verlag Berlin Heidelberg (2001) 33-49
18. Tellioğlu, H.: A Coordination Mechanism Based on Ontologies: Methodology and Implementation. VIP Scientific Forum of the International IPSI-2003 Conference, 5-10 October, Montenegro (2002)
19. Tolksdorf, R.: Coordinating Services in Open Distributed Systems with LAURA. First International Conference on Coordination Models, Languages and Applications (Coordination'96), 15-17 April (1996) 386-402