Coordination Theory and Software Process Technology

R. Mark Greenwood

Department of Computer Science
University of Manchester
Manchester M13 9PL, UK *
email: markg@cs.man.ac.uk

Abstract. Coordination theory is an interdisciplinary approach to studying the management of dependencies among activities. By its very nature software process technology deals with coordination. However it often expresses coordination in terms of low level details. An effective coordination theory would give us a better set of coordination abstractions. We illustrate the close relationship between these fields and propose areas where they could learn from each other.

1 Introduction

Coordination theory is the term used by Malone [9] to refer to the interdisciplinary study of coordination. It draws on a variety of different disciplines including computer science, organisation theory, management science, and economics. In these disciplines there are a variety of coordination problems and a range of solutions have been evolved to cope with them. One aim of coordination theory is to provide a language which can describe coordination problems and solutions in a way which allows them to be compared across disciplines.

Software Process Technology deals with coordination. By their very nature process centered software engineering environments coordinate their users and tools. Often, however, this coordination is achieved by low level details, the sending of individual messages. In general it is not possible to look at a process model and get an abstract view of its coordination dimension.

We will illustrate the close relationship between software process technology and coordination theory through an example. We hope that this will persuade the reader that the SPT community could both learn from, and contribute to, coordination theory. Finally, we make some tentative suggestions of benefits from understanding the relationship between these fields.

2 Coordination

Coordination is one of those ephemeral things which are best noticed by their absence. You spend a whole afternoon trying to find out who is supposed to

* previous address: Department of Electronics and Computer Science, University of Southampton, S017 1BJ, UK

be updating the `xyz` function with the extra parameter which you need, and eventually track down someone who says they thought that change was no longer required. This general feel is captured in Singh's definition: "Coordination is the integration and harmonious adjustment of individual work efforts towards the accomplishment of a larger goal" [10]. In [9] Malone proposes that:

> Coordination is managing dependencies between activities.

and one way of progressing coordination theory would be to classify different type of dependencies, and identify the coordination processes used to deal with them. The following initial list of dependency types is given [9].

- Shared resources (including task assignments)
- Producer/consumer relationships
- Simultaneity constraints
- Task/subtask

Example coordination processes range from economic markets (a process to allocate scarce resources) to the mutual exclusion techniques of computer science.

A number of coordination languages have been proposed: Role Activity Diagrams [8], Diplans [7] and Role Interaction Nets [10]. These have been used to model organisational processes. In comparison with most languages applied to the software process, their focus is slightly different. Most software process modelling languages focus mainly on activities and their temporal relationships. The coordination languages tend to emphasise the allocation of activities into roles within an organisation, and the corresponding coordination requirements which arise from this allocation.

3 An Example

In [7] the "3 parts, 2 assemblies" problem is used to introduce Diplans. We will use it to give examples of the dependency types mentioned above.

The "3 parts, 2 assemblies" problem deals with the development of the following product family:

- Assembly `aAC` with component `cA`, component `cC`
- Assembly `aBC` with component `cB`, component `cC`

From this product structure is is easy to conceive of a generic development process:

1. three activities `A`,`B` and `C` to develop the three components, `cA`,`cB` and `cC` respectively.
2. two activities `AC` and `BC` to develop the assemblies with `AC` dependent on `A` and `C`, `BC` on `B` and `C`.

In this example there is an obvious producer/consumer dependency between the components and their respective assemblies. The simplest way of managing this is to decide that the assembly activities will wait for the completion of the respective component activities. We will formalise this coordination process using a simple process modelling language *coPML* [4, 5]. In *coPML* a process is modelled in terms of events, a state, and a set of constraints on the state. Our simple model will have 10 events, $startX$ and $finishX$ for each activity `X`, and the state will have 10 counts, $\#EV$ for each event EV. Our coordination process is captured by the constraints: there are 5 basic constraints, which reflect that activities start, finish, and are not repeated, and 4 constraints to cope with the producer/consumer dependencies:

$$0 \leq \#finishX \leq \#startX \leq 1 \; where \; X = A, B, C, AC, BC$$
$$\#startAC \leq \#finishA \quad and \quad \#startAC \leq \#finishC$$
$$\#startBC \leq \#finishB \quad and \quad \#startCB \leq \#finishC$$

A simultaneity dependency might mean tasks A and C cannot be done at the same time. We can amend our process to ensure this:

$$\#startA - \#finishA + \#startC - \#finishC \leq 1$$

A shared resource dependency might arise because there are only 2 people to do the development. If `A` and `AC` are allocated to person `p1`, `B`, `C` and `BC` to person `p2`, then `B` and `C` compete for p2's time. In this case we might want to prioritise `C` over `B` to reduce the chance of A being kept waiting:

$$\#startB \leq \#finishC$$

If we formulate the problem in terms of goals then it falls naturally into task/subtask dependencies rather than producer consumer ones. The assemblies `aAC` and `aBC` need to be developed, for `aAC` we need to develop `cA` and `cC`, for `aBC` we need `cB` and `cC`. We can imagine a different coPML model with constraints such as $achieveAC \Rightarrow achieveA \wedge achieveC$.

There are some types of coordination which are harder to express. In developing assembly `aAC` we might want a more concurrent approach with activities `A` and `AC` being worked on together. In this scenario `AC` could be kept up to date with the current state of `cA`, and able to request alterations which make developing `aAC` easier.

4 Possible Benefits

The above example illustrates that the software process technology community could contribute to the development of coordination theory. There are a large number, and variety, of dependencies between activities in the software process. Process modelling languages provide ways of expressing the coordination processes which manage these dependencies, and process enactment systems provide examples of exploiting the coordination potential of state-of-the-art technology. If SPT has much to contribute, what are the potential benefits?

There is nothing so practical as a good theory. [6]

We currently lack good coordination abstractions. This means that we consider coordination at a low level and prefer coordination processes on the basis that they are easier to implement rather than more appropriate. A good coordination theory would not only encourage process modellers to express their coordination requirements, it would also promote the recognition and reuse of process fragments. It might also help to relate coordination requirements to the networking facilities used to satisfy them.

Coordination theory could also help in the comparison of process modelling languages and enactment systems. The small example above illustrates that some dependencies arise from the allocation of activities to actors which is only weakly represented in coPML. In [7] it is emphasised that coordination involves both the temporal and this "spatial" dimension. The "spatial" dimension is the co-locating of the entities needed to do the work and the agent or actor. In PSS [1, 2] this would be dealt with by how activities are mapped to PML roles, in Process WEAVER [3] by the work contexts.

Another benefit would be a better understanding of the relationships between software process technology and other coordination disciplines. This includes other areas of computer science such as Computer Supported Cooperative Working (CSCW) and groupware. In addition, the interdisciplinary nature of coordination theory would help to clarify whether the type and number of dependencies between software process activities was significantly different from those for other processes.

Acknowledgements

Thanks are due to colleagues at the Informatics Process Group at Manchester University and the Declarative Systems Laboratory at Southampton University for many useful discussions.

References

1. R.F. Bruynooghe, R.M. Greenwood, I. Robertson, J. Sa, R.A. Snowdon, and B.C. Warboys. PADM: Towards a total process modelling system. In A. Finklestein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 293–334. Research Studies Press, 1994.
2. R.F. Bruynooghe, J.M. Parker, and J.S. Rowles. PSS: A system for process enactment. In *Proceedings of the First International Conference on the Software Process*, pages 142–158, Redondo Beach, California USA, October 1991.
3. C. Fernström. Process WEAVER: Adding process support to UNIX. In *Proc of the Second International Conference on the Software Process*, pages 12–26, Berlin, Germany, February 1993. IEEE Computer Society Press. sponsored by Rocky Mountain Institute of Software Engineering.
4. R.M. Greenwood. *Modelling Processes with Constraints.* PhD thesis, University of Southampton, 1994. in preparation.

5. R.M. Greenwood. Modelling processes with constraints. In B.C. Warboys, editor, *Software Process Technology: Third European Workshop, EWSPT'94*, pages 167–170, Villard de Lans, France, February 1994. Springer-Verlag. Lecture Notes in Computer Science 772.
6. C. Handy. *Understanding Organisations*. Penguin Books, fourth edition, 1993.
7. A.W. Holt. Diplans: A new language for the study and implementation of coordination. *ACM Transactions on Office Information Systems*, 6(2):109–125, April 1988.
8. A.W. Holt, H.R. Ramsey, and J.D. Grimes. Coordination system technology as the basis for a programming environment. *Electrical Communication*, 57(4):308–314, 1983.
9. T.W. Malone and K. Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.
10. B. Singh. Interconnected roles (IR): A coordination model. Technical Report CT-084-92, MCC, 1992.

This article was processed using the LaTeX macro package with LLNCS style