Using Distributed Constraint Satisfaction to Build a Theory of Congruence

James Herbsleb, Anita Sarma
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
<jdh,antz>@cs.cmu.edu

Audris Mockus
Avaya Labs Research
Basking Ridge, NJ 07920
audris@avaya.com

Marcelo Cataldo
Research and Technology Center
Bosch Corporate Research
Pittsburgh, PA 15212, USA
marcelo.cataldo@us.bosch.com

ABSTRACT

Distributed Constraint Satisfaction Problem (DSCP) has been proposed as a methodology to frame and analyze coordination in software development. Here, we propose concrete ways to cast rich social and product dependence graphs of software projects into the DSCP framework, suggest how the lack of congruence among these graphs may affect primary software engineering outcomes, and discuss the DSCP machinery that is most likely to provide necessary tools to test these hypotheses.

1. INTRODUCTION

Coordination is a critical factor for the success or failure of a software project. Studies have shown that a lack of appropriate coordination is responsible for delays in project and cost overruns [3, 4]. Coordination in software development arise because of a number of reasons (e.g., interdependencies among artifacts, shared resources, and non uniform skill set among developers). Teams have to consider these issues to be able to coordinate among each other.

We have argued in prior work that coordination in software engineering is a distributed constraint satisfaction problem (DCSP) [5] in which engineering decisions form a constraint network. In this position paper, we argue that DCSP can be used to form a theoretical basis for congruence that lends clarity to the idea and generates specific, testable hypotheses.

2. CONSTRAINT SATISFACTION

The need for coordination arises from the core technical activity of making engineering decisions where the alternative chosen for any given decision potentially influences how one evaluates the choices for a number of other decisions. The full extent of these influences is often quite difficult to determine, hence decision-making is generally performed with some degree of uncertainty and with imperfect information.

We claim that the irreducible interdependence among tasks in software engineering is a distributed constraint satisfaction problem (DCSP), and consequently coordination in a development organization is the execution of an "algorithm" that solves the DCSP. In a DCSP, decisions are embedded in a network of constraints, and are potentially owned by many agents. Finding a solution then generally requires coordination among the agents. Given the boundedly rational nature of such decision-making, solutions will generally be satisficing rather than optimal [6]. As with many coordination problems, coordination in software engineering can be carried out in a great many ways, involving a variety of design methods, social processes, communication regimens, communication tools, and so forth. Yet the problem has an irreducible core – if incompatible decisions are made and constraints not satisfied, then either the software will not work properly or expensive rework will be required.

More specifically, we define DCSP by applying Yokoo's [7] formulation_to the software engineering domain. A software project consists of a large set of engineering decisions that must be taken in order to complete the project. Decisions are represented as n variables $x_1, x_2, \ldots, x_n$ whose values are taken from finite, discrete domains $M_1, M_2, \ldots, M_n$. Assigning a value to a variable represents taking the decision represented by that variable.

A project has a set of constraints that operate over the variables that represent the engineering decisions. Given an assignment of a value for some variable, the constraints serve to limit possible values that can be assigned to other variables. Formally, constraints $p_k(x_{k1}, x_{k2}, \ldots, x_{kn})$ can be represented as predicates defined on the Cartesian product $M_{k1} \times M_{k2} \times \ldots \times M_{kj}$. Successfully completing a project is equivalent to finding an assignment for all variables that satisfies all constraints. In order to represent which decisions participate in which constraints, we can construct a matrix C of dimension n decisions by k constraints, where a non-zero value C_{nk} means decision n participates in constraint k.

The product CC^T, where C^T is the transpose of C, is a square matrix D of dimension n, where a nonzero value D_{ab} indicates that decision a participates in at least one constraint with decision b. We call this matrix the dependency matrix. D can often be estimated directly from data, e.g., dependencies in code as measured by call graphs or logical dependencies, or dependencies in design as represented, e.g., by design structure matrices. Any of these can potentially be used to estimate the D matrix.

In development organizations, decisions are made by multiple agents. Let B be a matrix of dimension n by a, where n is the number of decisions and a is the number of agents. A nonzero entry B_{na} indicates that decision n belongs to agent a. B can be thought of as representing task assignments – which agents take which decisions.

Finally, a square interpersonal dependency matrix R of dimension a, the number of people involved in the project, can be constructed. A non-zero entry R_{ab} indicates that developer a is

performing work that shares dependencies with the work performed by *b*. *R* can be computed as BDB^T, where B^T is the transpose of *B*. *R* shows which agents own decisions that need to be coordinated for the particular task or tasks represented by *B*.

This section has argued that coordination in software development is a DCSP, and successful coordination is equivalent to finding a solution. In the next section, we describe some coordination networks representing mechanisms that organizations use for solving coordination DCSPs. *R* is particularly important in this theory, since coordination networks often represent coordination as person by person networks, which can be compared to *R*, in order to see if coordination behaviors, organizational structure, or other mechanisms are a good match for the coordination problem.

3. COORDINATION NETWORKS

In general, research on organizations claims that coordination can be achieved in three basic ways: "programming," communication, and shared representations. Coordination by "programming" refers to imposing rules, practices, or agreements than ensure that subsequent behavior is coordinated in certain respects. Included in this category would be things like plans, interfaces, and development processes. Coordination by shared representations means that agents construct and refer to things like documents and displays to coordinate work. In software, this could be things like project status data or test results. While we think the theory developed in this paper could readily be extended to these forms of coordination, we focus here on coordination by communication.

Agents attempt to solve a DCSP by making decisions, which we represent as assigning values to variables, and communicating with other agents. Communication behaviors differ in many ways, including what the agents communicate, when they communicate, and with whom they communicate. Patterns of communication emerge over time, and these patterns can lead to familiarity and enhanced ability to coordinate among frequently-communicating agents. These varieties of communication can be represented as social networks with different edge types.

Coordination networks that we have used in prior research [2] include communication over various channels, team membership, and geographic location. Each coordination network has people as nodes, and edges (possibly weighted) represent the type of coordination activity of interest, such as communication or communication proxy such as team membership. For computational purposes, each network can be represented as a square matrix *Q* of dimension *a*, where *a* is the number of people involved. This, of course, is the same dimension as interpersonal dependency matrix *R* in the previous section.

We have now described a way of representing the interpersonal dependency network that represents a fundamental property of the DCSP, and various coordination networks that impact how and how well organizations will solve the DCSP. In the next section, we describe the concept of congruence that brings *R* and *Q* together.

4. CONGRUENCE AND CONSTRAINTS

Congruence implies some form of correspondence of the interpersonal dependency network *R* and various coordination networks Q_m. There can obviously be many flavors of congruence, and such correspondences can be computed in a variety of ways. The underlying idea is simply that the various Q_m represent features of the development organization that will profoundly impact its ability to solve various DCSPs. The important relationships between *R* and Q_m may sometimes be simple, e.g., isomorphism or homomorphism, or they may be complex. The nature of the relations presents key research questions.

The fundamental propositions of this theory are:

P1: Higher levels of congruence predict more effective project coordination.

P2: As a consequence of P1, higher levels of congruence lead to better project outcomes.

P3: Decisions, particularly those made early in a project, can substantially alter the structure of *D* and *R*, thereby changing the nature of the coordination problem.

5. RESEARCH PROGRAM

Progress is needed along several fronts in order to complete the theory, test it empirically, and exploit the new knowledge to improve software development practice. Roughly we can divide the activities into two distinct groups: (1) the discovery phase, which would generate a variety of concepts and measures related to common coordination situations and relevant ways to measure congruence among them and (2) the application phase, which would use insights from case studies to propose practices and tools to improve software development.

5.1 Discovery

Connection of coordination networks and DCSP. It seems intuitively clear that various kinds of coordination networks, which represent key aspects of how an organization coordinates its work, will strongly influence the organization's ability to solve a DCSP of a particular form. Yet much theoretical and empirical work needs to be done to achieve a more precise understanding of this relationship. What is important, ultimately, is how the coordination networks shape the problem-solving behavior of the organization, and how effectively this problem-solving behavior addresses the particular DCSP representing the coordination problem.

Good measures of interpersonal dependency networks for a variety of tasks. One approach is to derive these networks from technical dependencies in code and other development artifacts. In code, for example, dependencies can be measured in terms of syntactic dependencies such as function calls or reading or writing data. It is not clear which of these measures can be used to produce the best estimates of dependency networks. Design structure matrices provide an appropriate example of a representation for decision-constraint networks for designs and architectures. It might also be possible to augment architecture design environments such as ArchE [1] to build the decision

constraint network from the properties of an architecture and a set of rules associated with quality attributes.

Identifying and measuring various kinds of coordination networks. There are potentially a large number of ways to coordinate engineering decisions, and finding appropriate measures for them is a challenge. Many can be computed from data often found in software archives, such as Bugzilla logs (e.g., who has communicated with whom about a bug) and version control systems (e.g., who has worked with whom in the past). Other potential coordination networks, such as knowledge networks and trust networks, that may impact coordination, may need to be measured by surveys or other means. Given the number and diversity of coordination networks, it may often be desirable to combine them, e.g., by means of matrix operations, to generate more precise or more general measures of coordination. Alternatively, selecting a subset of networks most salient to the phenomena under study may help solve more specific problems.

Computing congruence and its effects. As mentioned in the previous section, there are many degrees of freedom in the computation of congruence, including selection of decision-constraint network, selection of coordination network, and selection of how to compare them. Even more challenging is the development of theoretical refinements and performing empirical research to understand the various effects of different levels of congruence for development tasks. [2] provides an example of empirical analysis of Modification Request archives and communication archives to compute congruence measurements.

5.2 Applications

Collaborative and awareness tools based on congruence. Achieving an understanding of what kinds of coordination networks are important for resolving a particular type of decision-constraint network observed in actual projects will provide a basis for collaboration and awareness tools.

Application of formal methods. While we have no specific approach in mind, the formulation of coordination presented here allows graph-theoretic analysis of many aspects of coordination, which we hope will allow useful application of formal methods of evaluation. The novelty of application would be the potential of making the social component of software development amenable to formal methods.

Analysis of existing development methods. Many existing software engineering techniques are implicitly or explicitly designed to support solving a DSCP. For example, the daily standup meetings advocated in some flavors of agile methods are explicitly designed, at least in part, to facilitate coordination through frequent communication. Our theoretical framework may help highlight the exact role of each technique and to propose novel approaches that are not yet used.

Structure based on coordination needs. Given a better understanding of how architectural and design decisions impact coordination, work on congruence could provide a different way of thinking about task partitioning. Rather than basing modules and components purely on technical considerations, better choices could potentially be made by also taking account of the impact of design decisions on coordination requirements.

6. EXTENSIONS

In order to be more complete, and to predict and describe a wider range of coordination issues, the theory will need several extensions, including at least the following:

Time. Software development is dynamic, and as decisions are made over time, the nature of the coordination problem changes. If *m* decisions have been taken, then the DCSP consists only of *n-m* decisions and the constraints among them. The network is not only smaller, but it may also have very different network properties, depending on the location of the executed decisions in the original network. Removing the executed decisions may create a disconnected graph of more isolated components, for example. The coordination networks most appropriate for solving this reduced problem may be quite different from the networks that best matched the original DCSP. All of this assumes, of course, that decisions once made are never changed, which is an oversimplification. Backtracking happens often, but is expensive, so decisions once made tend not to get changed unless no solution can be found.

Path dependency analysis. Another potentially interesting area is path dependency analysis. The dependency matrix D described earlier represents the space of technical decisions and their relationships. As technical decisions are made, specific sub-parts of the decision space and their relationships could become critical or irrelevant. Then the model proposed here would provide the machinery to evaluate how specific sets of decisions impact subsequent ones and how that particular sequence of events relates to the coordination activity required by the organization.

What agents know. The agents' knowledge of decisions that have been made, the constraint predicates, and the ownership of decisions are highly likely to influence the effectiveness of various coordination networks to resolve DCSPs. For example, if an agent knows who owns a decision, and knows about a constraint affecting both that decision and one of his own decisions, then the agent knows who to communicate with to negotiate how to address the constraint, or to find out if a decision has already been made by the other agent. On the other hand, if the agent is not aware of the constraint, he may not know of any reason to communicate with that agent, suggesting that some other method such as unplanned communication or automated dependency detection may be required. Understanding what knowledge is needed in order to function effectively in the context of various coordination networks, and what kinds of communication and artifacts can provide that knowledge, are important research areas.

Artifacts. A more complete theory of coordination would include artifacts that agents use to coordinate. For example, agents may coordinate by examining changes in the code, design documents, or project status reports. Artifacts could perhaps be represented in bipartite graphs with agents and artifacts as nodes. Or they could be represented in additional matrices, e.g., agents by artifacts, or decisions by artifacts. In either case, this is an area rich in possibilities, since coordination by shared representations is one of the primary modes of coordination in general.

Prescribed behavior. Another area that should be explored in order to create a more complete theory is what we're calling prescribed behavior, and organizational theorists would call coordination by "programming." (For our field, we are using new terminology in order to avoid the obvious confusions.) When does it make sense to spend up front effort on a detailed architecture, defined process, plans, or requirements to coordinate the work, and when is a more agile approach superior?

7. DISCUSSION

Coordination is a fundamental issue in software engineering and in almost all kinds of group work. Many ways to coordinate work exist, and there are many drivers for the need to coordinate, they have largely been viewed as individual, isolated problems and only point solutions to some of these problems were offered. Our proposed theory provides the framework to integrate these diverse approaches into a coherent problem from which testable hypotheses can be derived and future progress facilitated.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge support by the National Science Foundation under Grants No. IIS-0414698, IIS-0534656, and by the Software Industry Center at Carnegie Mellon University and its sponsors, especially the Alfred P. Sloan Foundation. Our thanks to attendees of the IFIP WG 2.9 for helpful feedback on an earlier version of this paper.

9. REFERENCES

[1] Bachmann, F., Bass, L., and Klein, M., Preliminary Design of ArchE: A Software Architecture Design Assistant, CMU/SEI-2003-TR-021, Carnegie Mellon University, 2003,

[2] Cataldo, M., et al., *Identification of coordination requirements: implications for the Design of collaboration and awareness tools*, in *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*. 2006, ACM Press: Banff, Alberta, Canada.

[3] Herbsleb, J.D. and Grinter, R.E. Splitting the Organization and Integrating the Code: Conway's Law Revisited. in *21st International Conference on Software Engineering (ICSE 99)*. 1999. Los Angeles, CA: ACM Press.

[4] Herbsleb, J.D. and Mockus, A., An Empirical Study of Speed and Communication in Globally-Distributed Software Development. *IEEE Transactions on Software Engineering, 29*, 3 (2003), p. 1-14.

[5] Herbsleb, J.D., Mockus, A., and Roberts, J.A. Collaboration in Software Engineering Projects: A Theory of Coordination. in *International Conference on Information Systems*. 2006. Milwaukee, WI.

[6] Simon, H.A., *The sciences of the artificial*. Second ed. 1981, Cambridge, MA: The MIT Press.

[7] Yokoo, M., *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-agent Systems*. 2001, New York: Springer.