Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 108 (2004) 3–9
www.elsevier.com/locate/entcs

Time, Abstraction, Causality and Modularity in Interactive Systems

Extended Abstract

Manfred Broy[1]

Institut für Informatik, Technische Universität München
D-80290 München, Germany

Abstract

We study discrete models of interactive distributed systems structured into components and operating concurrently in a time frame. For such models of the data or signal flow in interactive system we assume that there is a source and a cause for each communication event and its associated information. To understand the logical dependencies for the events of systems causality is a key issue for reasoning about the event flow. Being interested in a structured modular approach we want to be able to abstract away all internal aspects of systems that are used as components within a system's architecture. We speak of interface abstraction. The interface abstraction is to keep only the aspects relevant for the usage of the component and the construction of the interface abstraction of the architecture. We speak of modularity if the interface abstraction of an architecture is the result of the composition of the interface abstractions of all its components. In particular, we discuss and study the relationship and dependencies between causality, input and output, compositionality, and the granularity of time.

Keywords: Interactive systems, time, composition, modularity, causality, input/output, abstraction

1 Fundamentals

We study the information flow in a system of components. Information flow evolves in physical or technical systems in a time frame. There are many ways to represent events and information (and on the contrary there are many

[1] Email:broy@in.tum.de

doi:10.1016/j.entcs.2004.11.003

ways to interpret certain phenomena as carrying information) that evolve in physical or technical systems.

In fact one way to look at this issue is to see causality as a principle that holds for any form of information processing systems independent of their technical or physical representation.

Of course, we can also model causality for event structures that model systems by concurrent traces. There causality is a logical relationship between the events and actions of a system.

We study a specific form of causality for system structured into families of components. Here we concentrate on the causality between input and output events. We claim that:

- there is a canonical notion of causality between input and output.

Causality is closely related to a model of time. A fundamental question concerns the model of time. Basically there are two essentially different models of time, discrete (digital) time and continuous (analog) time. A first critical question for system models has to do with time:

- Is it necessary to work with continuous time or is discrete time good enough?

Whenever two subsystems interact and thus mutually influence each other within a larger system we may interpret this as a form of information exchange. Often the information flows in both directions. This leads to a set of further questions:

- Can we model information exchange always as a directed process with a sender and a receiver (modeling mutual exchange as two steps of directed information exchange)?
- Do we need a model of information exchange that takes into account the states of two or more subsystems and calculates from that state the new state of the system?

We are interested basically in two issues:

- What are the universal laws of causality?
- How does causality and time relate?

The basic principle of causality in information flow of components with input and output is as simple as that (here we assume that a system cannot predict its future input and thus there is no anticipating gate no gate that can predict its future input)

- Information (input) can only be processed and forwarded as soon as/after it had been received.

We discuss the notion of strong and that of weak causality. In strong causality the time model strictly separates all events that may be causal for other events from those that are causal for them.

We consider a system in several time granularities. We assume the following principles:

- If we choose the time granularity fine enough then
  - If the time scale of the input is fine enough then there is no non-determinism/underspecification left in the system due to missing information about the input timing.
  - If the time scale is finer than the minimal delay, then the behavior function is *strongly causal*.
  - Every system behavior is weakly causal. Strong causality may be abstracted away by too coarse time models.
- Every system behavior is implicitly strongly causal. This means there exists a strongly causal behavior such that the system behavior is a time coarsening.

If we choose the time granularity not fine enough then we get a system behavior that is only weakly causal but implicitly strongly causal.

2 Causality in Interface Abstractions

For input/output information processing devices there is a crucial notion of causality. Certain output depends causally on certain input. Causality indicates dependencies between the actions of information exchange of a system. So far interface behaviors are nothing but relations represented by set valued functions. In the following we introduce and discuss the notion of causality for interface behaviors.

Interface behaviors generate their output and consume their input in a time frame. This time frame is useful to characterize causality between input and output. Output that depends causally on certain input cannot be generated before this input has been received.

Nevertheless, proper timing permits instantaneous reaction [8]: the output at time t may depend on the input at time t. This may lead into problems with causality if we consider in addition delay free feedback loops. To avoid these problems we strengthen the concept of proper time flow to the notion of causality.

If F is time guarded then the output in the t-th time interval does not depend on input that is received after the (t-1)-th time interval. Then F is properly timed and in addition reacts to input received in the (t-1)-th time

interval not before the t-th time interval. This way causality between input and output is guaranteed and explicitly visible according to the fine time granularity.

3 Causality in Composition

When composing components systems causality proves crucial. Each component acts and reacts. Which if each steps are proactive and happen independent of the existence of steps of other components and which happen only in response ("reaction") to steps of other components can only be determined by the logic of causality of a component.

If we do not have the full notion of causality in the interface abstraction of a component we cannot determine which events will definitely happen, which events will happen in reaction to those and which will not take place due to causality properties. Then it may become impossible to exclude behaviors the represent so-called causal loops and therefore are operationally impossible. In particular, the dead lock properties cannot be deduced correctly from the interface abstraction of system components.

4 Coarsening the Time Scale

To make for a component the time scale coarser by the factor n is a form of abstraction. This coarsening may introduce some kind of nondeterminism and underspecification due to the coarser time scale. In particular we are interested in the problem of loosing causality information by such a time coarsening.

A special case is to get rid of all time information. If the output of a component is depending on the timing of the input a behavior generated by full time abstraction shows a lot of nondeterminism. However, if the output produced does not depend on the timing of the input messages but only on their values and the order in which they arrive behavior generated by full time abstraction will rather be more deterministic. However causality information may be lost that way.

If a component is weakly causal its time abstraction by a finite factor n is weakly casual, too. However, strong causality is not maintained.

5 A Formal Approach to Causality

As formulated above, we assume that for each real system behavior there is a time scale that is fine enough to capture all time differences especially for the

delay between input and output. In such a time scale the behavior is strongly causal.

For a feedback loop being the result of a component composition a behavioral fixpoint is causal, if all output is causal by some input.

Strong causality has a number of advantages since it makes the reasoning about systems more concrete and simpler. In particular, it is easy to treat feedback loops by fixpoints for strongly causal behaviors since time guardedness guarantees for instance unique fixpoints for deterministic functions. In other words, for strong causality all fixpoints are causal.

The disadvantage of time guardedness is its limited abstractness illustrated by the fact that in sequential composition the delays accumulate. As a result composition is not as abstract as needed. A composed system has always delays larger than two. This difficulty is avoided by weak causality. But then the reasoning about fixpoints gets more involved. There exist fixpoints that are not causal.

This shows that the assumption of causality leads to proof principles for systems.

References

[1] M. Abadi, L. Lamport. The Existence of Refinement Mappings. Digital Systems Research Center, SRC Report 29, August 1988.

[2] M. Abadi, L. Lamport. Composing Specifications. Digital Systems Research Center, SRC Report 66, October 1990.

[3] L. Aceto, M. Hennessy. Adding Action Refinement to a Finite Process Algebra In *Proc. ICALP 91*, Lecture Notes in Computer Science 510, 1991, 506-519.

[4] P. Andrews. An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof. Computer Science and Applied Mathematics. Academic Press 1986.

[5] R.J.R. Back. Refinement Calculus, Part I: Sequential Nondeterministic Programs. REX Workshop. In: J. W. deBakker, W.-P. deRoever, G. Rozenberg (eds): Stepwise Refinement of Distributed Systems. Lecture Notes in Computer Science 430, 1989, 42-66.

[6] R.J.R. Back. Refinement Calculus, Part II: Parallel and Reactive Programs. REX Workshop. In: J. W. de Bakker, W.-P. de Roever, G. Rozenberg (eds): Stepwise Refinement of Distributed Systems. Lecture Notes in Computer Science 430, 1989, 67-93.

[7] L. Bass, P. Clements, R. Kazman. Software Architecture in Practice. Reading, MA, Addison-Wesley 1998.

[8] G. Berry, G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. INRIA, Research Report 842, 1988.

[9] G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language for Object-Oriented Development. Version 1.0. RATIONAL Software Cooperation.

[10] R. Breu, R. Grosu, F. Huber, B. Rumpe, W. Schwerin. Towards a Precise Semantics for Object-Oriented Modeling Techniques. In: H. Kilov, B. Rumpe (eds.): Proceedings ECOOP'97 Workshop on Precise Semantics for Object-Oriented Modeling Techniques, 1997, Also: Technische Universität München, Institut für Informatik, TUM-I9725, 1997.

[11] J. D. Brock, W. B. Ackermann. Scenarios: A Model of Nondeterminate Computation. In: J. Diaz, I. Ramos (eds): Lecture Notes in Computer Science 107, Springer 1981, 225-259.

[12] M. Broy. Mathematical System Models as a Basis of Software Engineering. J. van Leeuwen (ed.): Computer Science Today. Lecture Notes of Computer Science 1000, 1995, 292-306.

[13] M. Broy. Algebraic Specification of Reactive Systems. M. Nivat, M. Wirsing (eds): Algebraic Methodology and Software Technology. 5th International Conference, AMAST 96, Lecture Notes of Computer Science 1101, Heidelberg, Springer 1996, 487-503.

[14] M. Broy. The Specification of System Components by State Transition Diagrams. Technische Universität München, Institut für Informatik, TUM-I9729, May 1997.

[15] M. Broy. Towards a Mathematical Concept of a Component and its Use. First Components' User Conference, Munich 1996. Revised version in: Software–Concepts and Tools 18, 1997, 137-148.

[16] M. Broy. Refinement of Time. M. Bertran, Th. Rus (eds.): Transformation-Based Reactive System Development. ARTS'97, Mallorca 1997. Lecture Notes in Computer Science 1231, 1997, 44-63, To appear in TCS.

[17] M. Broy. Compositional Refinement of Interactive Systems. Digital Systems Research Center, SRC Report 89, July 1992, Also in: Journal of the ACM, Volume 44, No. 6 (Nov. 1997), 850-891.

[18] M. Broy. From States to Histories. In: D. Bert, Ch. Choppy, P. Mosses (eds.): Recent trends in Algebraic Development Techniques. WADT'99, Lecture Notes in Computer Science 1827, Springer 2000, 22-36.

[19] M. Broy, M. Breitling, B. Schätz, K. Spies: Summary of Case Studies in Focus - Part II. Technische Universität München, Institut für Informatik, SFB-Bericht Nr. 342/40/97A, 1997.

[20] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, R. Weber. The Design of Distributed Systems - An Introduction to Focus. Technische Universität München, Institut für Informatik, Sonderforschungsbereich 342: Methoden und Werkzeuge für die Nutzung paralleler Architekturen TUM-I9202, January 1992.

[21] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, R. Weber. Summary of Case Studies in Focus - a Design Method for Distributed Systems. Technische Universität München, Institut für Informatik, Sonderforschungsbereich 342: Methoden und Werkzeuge für die Nutzung paralleler Architekturen TUM-I9203, January 1992.

[22] M. Broy, B. Möller, P. Pepper, M. Wirsing. Algebraic Implementations Preserve Program Correctness. Science of Computer Programming 8 (1986), 1-19.

[23] M. Broy, K. Stølen. Focus on System Development. Springer 2001 (to appear).

[24] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns. New York: Wiley 1996.

[25] K. M. Chandy, J. Misra. Parallel Program Design: A Foundation. Addison Wesley 1988.

[26] J. Coenen, W.P. deRoever, J. Zwiers. Assertional Data Reification Proofs: Survey and Perspective. Christian-Albrechts-Universität Kiel, Institut für Informatik und praktische Mathematik, Bericht Nr. 9106, Februar 1991.

[27] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley 1995.

[28] C.A.R. Hoare. Proofs of Correctness of Data Representations. Acta Informatica 1, 1972, 271-281.

[29] C.A.R. Hoare. Communicating Sequential Processes. Prentice Hall, 1985.

[30] F. Huber, B. Schätz, G. Einert. Consistent Graphical Specification of Distributed Systems. In: J. Fitzgerald, C. B. Jones, P. Lucas (ed.): FME '97: 4th International Symposium of Formal Methods Europe, Lecture Notes in Computer Science 1313, 1997, 122-141.

[31] G. Kahn. The Semantics of a Simple Language for Parallel Processing. In: J.L. Rosenfeld (ed.): Information Processing 74. Proc. of the IFIP Congress 74, Amsterdam: North Holland 1974, 471-475.

[32] G.T. Leavens, M. Sitaraman. Foundations of Component-Based Systems. Cambridge University Press 2000.

[33] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, W. Mann. Specification and Analysis of System Architecture Using Rapide. IEEE Transactions on Software Engineering, Special Issue on Software Architecture, 21(4): 336-355, April 1995.

[34] N. A. Lynch, E. W. Stark. A Proof of the Kahn Principle for Input/Output Automata. Information and Computation 82(1): 81-92 (1989).

[35] M. Moriconi, X. Qian, R. A. Riemenschneider. Correct Architecture Refinement. IEEE Transactions on Software Engineering, Special Issue on Software Architecture, 21(4): 356-372, April 1995.

[36] R. Milner. A Calculus of Communicating Systems. Lecture Notes in Computer Science 92, Springer 1980.

[37] B. Möller. Algebraic Structures for Program Calculation. In: M. Broy, R. Steinbrüggen (eds.): Calculational System Design. Marktoberdorf Summer School 1998. Nato Science Series, Series F: Computer and System Sciences 173, IOS Press: Amsterdam 1999, 25-100.

[38] B. Rumpe. Formale Methodik des Entwurfs verteilter objektorientierter Systeme. Dissertation, Technische Universität München, Fakultät für Informatik 1996. Published by Herbert Utz Verlag

[39] M. Shaw, P. Clements. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. Proceedings of the COMPSAC, Washington, D.C., August 1997.

[40] M. Shaw, D. Garlan. Software Architecture: Perspectives of an Emerging Discipline. Englewood Cliffs, NJ: Prentice Hall 1996.

[41] Specification and Description Language (SDL), Recommendation Z.100. Technical report, CCITT, 1988.

[42] J. M. Spivey. Understanding Z - A Specification Language and Its Formal Semantics. Cambridge Tracts in Theoretical Computer Science 3, Cambridge University Press 1988.

[43] Clemens Szyperski. Component Software - Beyond Object-Oriented Programming. Addison-Wesley, ACM Press 1998.