Aspects - Composing CSCW Applications.

Stephen Barrett and Brendan Tangney

Distributed Systems Group, Department of Computer Science, Trinity College Dublin, Ireland.

E-mail: {Stephen.Barrett,Brendan.Tangney}@dsg.cs.tcd.ie
Fax: +353-1-6772204

ABSTRACT: *Current approaches to CSCW application support are limited by their failure to support application distribution, internal application concurrency, anonymous communication, easy application integration, and run-time application behaviour modification. This paper argues that these limitations may be addressed at the language level. Accordingly, we introduce a new model and language which assimilates CSCW requirements as features tailored to the construction of open systems.*

KEY WORDS: *CSCW, Composition*

1 Introduction

"Every interesting concurrent system is built from independent agents which communicate"

[Milner, 1992]

COMPUTER Supported Cooperative Work is the field concerned with how people collaborate on any given task, and how computer applications, called groupware applications, may be developed to augment and better facilitate this process. It reflects a change in emphasis from the traditional use of the computer to solve problems to using the computer to facilitate human interaction [Ellis et al., 1991].

Researchers in the field have begun to explore how one might provide generic support for a wide range of groupware. This research has followed two threads which may be characterised as extending either current *language models* or current *operating system support* with functionality tailored to the needs of groupware applications. For example, Dourish [Dourish, 1994] has explored the applicability of reflection in toolkit design. His approach simplifies the construction of **specific classes** of groupware supported by his toolkits. He reports that adding reflection to the programming language via a reflective toolkit facilitates the building of groupware applications that are *open to change*. For example, a data management subsystem of a groupware application may be required to operate in a centralised or decentralised manner and potentially switch between the two at run-time. Dourish argues that using reflection in a CSCW toolkit is an appropriate mechanism for meeting this requirement. The alternative and more generic approach advocated by [Benford and Mariani, 1993], is to provide run-time support services separate from the application. Examples of services required by groupware include data sharing managers, vote managers, event managers and organisational databases. Research in this area is still in its infancy, with success being principally in the area of shared databases. This provides some support for application inter-working through shared data services.

Through a synthesis of CSCW literature (see section 4) we have identified a set of fundamental CSCW support requirements for which no existing solution provides full support. They are:

application distribution, (internal application) concurrency, anonymous communication, easy application integration, and *run-time behaviour modification.* This paper argues that these requirements may be usefully supported at the language level as features of a new object model. In particular it is argued that groupware applications required to inter-work with other applications are best constructed as **open systems**. By this we mean that they are best constructed as recomposable compositions of collaborating objects [Nierstrasz, 1995].

The open systems approach is impeded by limitations of current object-oriented and other programming techniques, in particular their failure to support *object composition* [Nierstrasz, 1995]. For example, abstract data types provide no means of expressing inter-object behavioural relationships [Sullivan and Notkin, 1992]. These relationships lay buried in the code of methods. Thus the object oriented approach, while having many advantages, does not allow us to specify the behaviour of groups of objects; it does not support *composition.*

Of the CSCW requirements we have identified, an object model which supports application *distribution, concurrency*, and *anonymous communication* is a sufficient platform on which to build a new open system model specifically tailored to the support of groupware. We are currently engaged in the design and implementation of this model, called the **Aspect** model. It in turn is based on a distributed event based object model known as ECO [Starovic et al., 1995]. It extends ECO to support inter-working and run-time modification of groupware applications.

The layout of the rest of this short paper is as follows. The following section and section 3 outline our model and language. Section 4 briefly discusses related work. Section 5 presents a summary of our approach. Finally in section 6 we report on the status of our current work and outline our future plans.

2 The Aspect Model

The distinctive feature of ECO is that it supports anonymous communication (*asynchronous* implicit invocations), a decoupling of event raising from event handler invocations[1]. ECO objects do not reference other ECO objects. Instead, an underlying run-time is responsible for distribution of events generated by ECO objects.

The aspect model is an object oriented composition model. An aspect is a specification detailing how a *composite component* may be instantiated by composing a set of *sub-components.* An aspect specification includes some type information (called a **slot** and expressed through an extended type model) describing each sub-component needed in the composite component. Any ECO object whose type conforms to a slot in an aspect may be used as a sub-component in that aspect. In our model, all objects are augmented by a extended type description, called a **shape**, which is used to determine conformance.

A good analogy for the aspect model is a child's jigsaw of the sort that has a frame into which pieces may be slotted. The pieces correspond to sub-components. Each has a specific shape which correspond to the sub-components type. The frame of the puzzle corresponds to an aspect. It has slots with specific shapes that pieces must conform to if they are to be slotted in. Any piece may be fitted into a specific slot if its shape conforms to the slot irrespective of whether the piece was originally made for that jigsaw or not. We say then that the sub-component **fills** the slot. The new piece need only conform to the shape. It may, for example be a different colour or material.

An aspect specification is implemented by an **aspect object**, which in effect implements an alternative to the default ECO run-time. It filters the communications both between sub-components under its control and between sub-components and entities external to the composition. The filtering policy implemented is specified in the aspect specification. To instantiate a composite component (which we generally refer to as an **aspect instance**), one places a set of sub-components under the control of an aspect object. We characterise our model as a two level computation in

[1]This is a more powerful model than the synchronous implicit model of method invocation via indirection. If *required*, synchronisity may be constructed above an event model.

which the communications of a base level of sub-components is reified by a *meta-level* aspect object.

The aspect model is recursive. In addition to slots describing sub-components, an aspect specification includes an extended type description (the frame itself has a shape). An aspect instance may therefore be used as a sub-component in another aspect. In this way, the aspect model provides a means of instantiating an aspect as a composition of sub-aspects[2]. Ultimately recursion in the model bottoms out to a collection of ECO objects (each augmented with a shape).

An aspect specifies a filtering policy for both external events arriving at an aspect instance, and for events generated by its sub-components. This is done at two levels. First, the aspect specifies the set of events that may be exported from the aspect and the set of events that may received by the aspect. Second, an aspect's slot descriptions specify, for each sub-component, the set of events each sub-component may generate and the set of events each may receive. This specification is imposed on a sub-component filling a slot. An event arriving at an aspect from some external entity will be propagated only to those sub-components filling slots whose description includes the event as an incoming event. An event generated by some sub-component will be propagated by the aspect (object) to peer sub-components according to the same criteria. However, if the event is of a type that may be exported to entities outside the aspect, then the aspect (object) will in addition export that event. In this way, a sub-component contributes to the functionality of the composite component both by generating events propagated externally, and by handling events delegated to it by the aspect object.

The aspect run-time allows sub-components to be dynamically replaced via a dynamic type conformance check performed by the aspect object. An application may thus be tailored dynamically to its possibly heterogeneous run-time environment by replacing unsuitable sub-components with others designed to integrate with local applications, system services or hardware configurations. More radical functional modification may be achieved in an application by selectively replacing an application's aspect objects with others which implement alternative aspect specifications. One may, for example, add new slots to the aspect or even new components and functionality to the application. Through aspects, our model provides definite points of composition at which the structure of an application may be redefined. At any point, the scope for transformation of a sub-component is limited only by the slot it fills.

3 The Aspect Language

We are currently developing a semantic model for the definition of aspects as linguistic constructs separate from the instances of abstract data types they compose. We have adopted the semantic approach of [Helm et al., 1990], namely the expression of composition separately and distinctly from class definition, over the alternative of modeling behavioural relationships as classes [Sullivan and Notkin, 1991]. We have identified a number of open issues in the design of a language supporting our model. Chief among these is development of a type theory for plug compatibility of aspects [Nierstrasz and Papathomas, 1990].

Type conformance to slots is the only limitation placed on a sub-components use. Sub-components may, for example, originate from unrelated code bases. Type information for each sub-component is required beyond compile time that describes its behaviour. Signatures do not provide enough information about the behaviour of an object to determine if it is safe to use in a new context [Nierstrasz and Papathomas, 1990]. Some information will be needed about how it appears to behave externally when exposed to particular events. Furthermore, ECO objects may dynamically modify their interface (through subscription and unsubscription of events). To extend our analogy, an sub-component's shape may vary over time. This complicates the description of an object's type to include state. The relationship between sub-component and slot is not that of simple sub-typing. Though simple sub-typing may suggest conformance between a sub-component and

[2] In this sense, an aspect is an equivalent abstraction to a subsystem of an object-oriented framework.

a slot, the restrictions placed upon a sub-component by a slot may result in that sub-component exhibiting incorrect behaviour.

4 Related Work

The underlying approach to events employed in ECO is very similar in spirit to that employed in many preceding CSCW systems, the subscription based active e-mail system *Khronika* being a good example [Lovstrand, 1991].

CSCW research has identified a need for increased flexibility in the construction and behaviour of groupware applications. The focus of much research is moving from monolithic application architectures to applications composed of independent but communicating computational units. Systems developed using this latter approach have been demonstrated to be more flexible and dynamically configurable. Examples of such systems and research efforts include the Btron2 Window system[Koshizuka, 1994], rendezvous [Hill et al., 1993] and ODP [Roseman and Greenberg, 1993]. However, these systems do not provide support for all the CSCW requirements we have identified.

Underlying collective support is advocated in [Benford and Mariani, 1993]. However, our approach differs in that we feel that supporting CSCW requirements is best achieved by supporting an appropriate object model and programming paradigm for CSCW rather than try to overcome the inadequacies of existing ones.

We have drawn on research in the field of composition [Kaiser and Garlan, 1987, Nierstrasz, 1995, Helm et al., 1990] in the development of our model. However, though compositional models of systems such as BETA [Knudsen et al., 1993] were found to be similar to ours, there are important differences. Our concept of composition seems more flexible as our aspects are truly independent computational modules whereas existing research into composition has focussed on composition as purely a language construct integrated at compile time. In our model an application is a redefinable set of independent but communicating aspects. This application model is supported by Parnas [Parnas, 1979] who argues that the presence or absence of an application component should be transparent to the other components.

5 Summary

An application is traditionally constructed by compiling code to produce an executable binary. This binary may, during execution, create objects which persist beyond the execution of the application. It may create many processes and may even migrate execution to other machines in a distributed system. It is still one program however: one binary. The aspect model is quite different from this. An aspect based system is constructed by manipulating the interactions of numerous stand-alone components. Interactions between components are anonymous: they simply act on events of unknown origin delivered by the aspect runtime and generate and pass new events to the runtime. By managing the propagation of events, an aspect runtime can coordinate a collection of components to produce application like behaviour. An aspect system can be viewed as application but its nature, as we have illustrated, is quite different from that of a traditional application. The boundary defining the set of interacting components is flexible. We can replace or add components. We can redesign the way its elements interact. We can even subsume the set in a larger set by establishing communication links between elements of it and other aspect systems. Whether we call the resulting system an application or a collection of applications is not clear, and therefore not very meaningful. If an aspect system is to be called an application then it is a very loose use of the word.

6 Current status

At time of writing, we have developed a linguistic approach broadly suitable to our Aspect model which we are using to test our ideas. We are currently elaborating both this linguistic approach and our model. Our medium term goals are the development of a type model compatible with our requirements, the definition of a prototype language built on this, and compiler and run-time support.

Acknowledgements

The authors would like to acknowledge the contributions of Stephen Brown and Brian Breathnach.

References

[Benford and Mariani, 1993] Benford, S. and Mariani, J., editors (1993). *Requirements and Metaphors of Shared Interaction.* Lancaster University. Esprit Basic Research project 6225, D4.1.

[Dourish, 1994] Dourish, P. (1994). Designing for change: Reflective metalevel architectures for deep customisation in cscw. Technical report, Rank Xerox EuroParc, Cambridge, UK.

[Ellis et al., 1991] Ellis, C., Gibbs, S., and Rein, G. (1991). Groupware,some issues and experiences. *Communications of the ACM*, 34(1).

[Helm et al., 1990] Helm, R., Holland, I. M., and Gangopadhyay, D. (1990). Contracts: Specifying behavioral compositions in object-oriented. In *Conference on Object-Oriented Programming: Systems, Languages, an Applications. European Conference on Object-Oriented Programming*, pages 169–180. ECOOP/OOPSLA, ACM Press.

[Hill et al., 1993] Hill, R. D., Brinck, T., Patterson, J. F., Rohall, S. L., and Wilner, W. T. (1993). The rendezvous language and architecture. *Communication of the ACM*, 36(1):63–67.

[Kaiser and Garlan, 1987] Kaiser, G. E. and Garlan, D. (1987). Melding software systems from reusable building blocks. *IEEE Software*, pages 17–24.

[Knudsen et al., 1993] Knudsen, J. L., Lofgren, M., Madsen, O. L., and Magnusson, B. (1993). *Object Oriented Environments, The Mjolner Approach.* The Object-Oriented Series. Prentice Hall.

[Koshizuka, 1994] Koshizuka, N. (1994). *Btron2 Window System: A window System Facilitating Cooperation among GUI Applications in Distributed Environments.* PhD thesis, Department of Information Science, Faculty of Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, 113 Japan.

[Lovstrand, 1991] Lovstrand, L. (1991). Being selectively aware with the khronika system. In *Proceedings of the European Conference on Computer Supported Collaborative Work (ECSCW)*, Amsterdam.

[Milner, 1992] Milner, R. (1992). *A calculus of communicating systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin Heidelberg NewYork.

[Nierstrasz, 1995] Nierstrasz, O. (1995). Requirements for a composition language. In *Proceedings of the ECOOP 94 workshop on Models and Languages for Coordination and Parallelism and Distribution*, LNCS. Springer Verlag.

[Nierstrasz and Papathomas, 1990] Nierstrasz, O. and Papathomas, M. (1990). Towards a type theory for active objects. In *Conference on Object-Oriented Programming: Systems,Languages, and Applications/European Conference on Object-Oriented Programming*. OOPSLA/ECOOP, ACM Press.

[Parnas, 1979] Parnas, D. L. (1979). Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, SE-5(2):128–137.

[Rodden and Blair, 1992] Rodden, T. and Blair, G. (1992). Distributed systems support for computer supported cooperative work. *Computer Communications*, 15(8).

[Roseman and Greenberg, 1993] Roseman, M. and Greenberg, S. (1993). Building flexible groupware through open protocols. In *Conference on Office Information Systems*, pages 279–288. ACM.

[Starovic et al., 1995] Starovic, G., Cahill, V., and Tangney, B. (1995). An event based object model for distributed programming. This conference.

[Sullivan and Notkin, 1991] Sullivan, K. J. and Notkin, D. (1991). Behavioural relationships in object-oriented analysis. Technical Report 91-09-03, Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 USA.

[Sullivan and Notkin, 1992] Sullivan, K. J. and Notkin, D. (1992). Behavioural relationships. Technical Report 92-03-08, Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195 USA.