Coupling Interaction Specification with Functionality Description

A. Kameas[1] S. Papadimitriou[1,2] G. Pavlides[1,2,3]

[1] Dept. of Computer Engineering and Informatics, Univ. of Patras, Patras 26110, Greece
[2] Computer Technology Institute, 3 Kolokotroni st., Patras 26221, Greece
[3] INTRASOFT SA, 2 Adrianiou st, Athens 12515, Greece

Abstract

In this paper, the solution used in the context of SEPDS (a Software Development Environment) to the problem of combining interactive behavior specification with functionality description of a distributed interactive application is presented. This solution consists of combining two specification models: IDFG to describe the interactive aspects of applications developed with the system and EDFG to describe their functionality. Both these models are data flow graph based and can be classified as process models. They use "actors" to represent performers of processes and "links" to represent data buffering and exchange, as well as roles and different perspectives. Although the two models have many semantical differences, they also have many common properties, that is why they can be straightforwardly combined in a process that enables designers think in users terms. To this end, action actors are used to represent the functions supported by the application, and context actors to represent the application user interface functions. In addition, links are used to represent the events that take place in the system (these may be user or system actions), the effects that these have on the screen, the context into which these take place and the goals that may be achieved using the application. Furthermore, the reusability and prototyping tools of SEPDS can be used to construct and test the application design.

1 Introduction

The need to build increasingly complex software systems has led in the development of SDEs (Software Development Environments) [6, 9], which not only provide assistance in software development, but also guarantee a standard level of quality, as they progressively integrate tools that support more phases of the software development process. With the evolution of technology, the need to build highly interactive applications that address non–computer expert users has recently come up. Despite the attempts, however, there still exists a gap between the designers' and users' model of an interactive application, due mainly to the unsuitability of the traditional application development techniques for the specification of interaction and the construction of user interfaces [11], and to the difficulty of combining an interaction model with an application data model [1].

In this paper, a solution that has been applied in the context of SEPDS, (Software Environment for the Prototyping of Distributed Systems) system [9], which is an engineering framework for the prototyping of distributed systems, is presented. This solution consists of using two distinct specification models: EDFG (Extended Data Flow Graph) for the specification of application functionality, and IDFG (Interactive Data Flow Graph) for the specification of the interactive behavior of this application. Both these models are based on Data Flow Graphs (DFG), and therefore, can be straightforwardly combined. In addition, they are used in much the same way, and as a consequence, designers that use SEPDS do not have to use additional effort to learn two different specification languages, while the prototyping subsystem of SEPDS is able to produce highly–interactive application prototypes.

EDFG and IDFG belong to the class of process models [4]; more specifically, they can be proved equivalent to Petri–Net models. Process models, in general, use the notion of a "process" (which is a set of partially ordered steps called "process elements") to represent an action towards some "goal". In both EDFG and IDFG, actors form the performers of process elements (the notion of agents is generally used in process models), by representing computations of arbitrary complexity. Links are used as a form of role representation (that is, to determine which set of process elements will be executed), as well as for the transportation and buffering of data, and the representation of screen effects that computations may have. Thus, these models represent explicitly the functional (i.e. what process elements are being executed and what flows of information are used by them), behavioral (i.e. conditions for the execution of process elements and how these interact) and organizational (i.e. where and by whom the execution of process elements takes place) perspectives of an application, while they may be used (that is, they include the appropriate constructs) for the representation of informational (i.e. explicit representation of the informational products consumed or produced by a process) perspective as well.

Process models usually distinguish between two enactors of a sequence of processes: users (in this case, the sequence is called a "script") and system (in this case, a "program"). This distinction is also used in this case, by allowing the modeling of both user and system actions as actor execution enablers. In addition, an explicit representation of goal–subgoal decomposition of user actions is used as the means to determine permitted states that may be reached from the current state. In this way, designers are "forced" to create an application that will be functioning close to the way users think when using it.

In SEPDS, EDFG serves as the data model, while IDFG is the event model. In the next two sections, both models are briefly presented, and subsequently the paper focuses on how these models are combined, by presenting their common and distinct properties and a simple example.

2 EDFG: The Data Model

The EDFG model [9] is a trade–off between formal models (e.g. [7]) and icon oriented models (e.g. [3]). According to the EDFG approach, a distributed application is described as a set of communicating graphs, with several communication primitives defined to support this communication.

An EDFG is a bipartite graph consisting of a set L of links and a set A of actors. Actors represent the computational components, while links are used for data buffering and transportation. There also exists a set E of directed arcs which connect actors to links and links to actors. A link gives input to an actor if a directed arc exists from the link to that actor; a link receives output from an actor if a directed arc exists from that actor to the link. The links can contain

tokens, which are value structures of arbitrary complexity. The number of tokens contained in all links at a given instant may be used to define the state of the EDFG, which is also called "EDFG marking".

An actor a is defined as a 7–tuple consisting of:

- *the set of input links* $IFS(a)$
- *the set of output links* $OFS(a)$
- *a precondition function* $PRE(a)$, which gives the conditions that must hold (that is, which of IFS links must contain tokens) for an actor to be executed
- *a postcondition function* $POST(a)$, which gives the conditions that result from the execution of the actor (that is, which of OFS links will receive tokens)
- *a function* $FUN(a)$, which represents the computation performed by the actor
- *a descriptor* $TYPE(a)$, which describes the level of actor complexity, and
- *a descriptor* $\delta(a)$, which refers to the execution time of the actor

Links are typed: a link type describes a set of actor links that are allowed to contain tokens of the same data type. When the designer constructs an EDFG, all the components of an actor (with the possible exception of $\delta(a)$) must be defined. Then links may be defined, with the system performing consistency and type–matching checks.

An actor is executed ("fires") when a subset of its input links defined by its PRE function contains tokens; all these tokens are then removed from the input links, while tokens are placed in the output links that are specified by the actor $POST$ function. Each actor firing modifies the distribution of tokens on links and thus produces a new EDFG marking (a new state).

3 IDFG: The Interaction Specification Model

IDFG combines features from both state–based (e.g. [5]) and user–oriented interaction models (e.g. [2]). It supports the specification of all interactive features of an application in a non–technical way and permits the construction of specialized interactive applications. To achieve this, description of the elements of the screen has been separated from the specification of actions that may be performed with these elements, leading to a more user–centered design.

Each IDFG [8] is also a bipartite graph. Nodes are of two different types: links and actors. Actors represent the actions that the system offers to its users and the goals they may achieve, while links are used to describe conditions. Directed arcs connect actors to links and links to actors. Actor firing and actor marking notions are also used in IDFG as they have been defined (although in the following, an equivalent definition of state will be presented).

Each actor consists of two parts: the behavioural part, which is made up of rules, and the functional part, which contains code segments. For every rule of the behavioural part, there exists a set of left–hand–side conditions that must hold for it to fire (the PRE function of the actor), and a set of right–hand side conditions that result from the firing (its $POST$ function). The set of the left–hand–side conditions of all links make up the IFS of the actor, while the set

of the right–hand–side conditions of all links make up its OFS function. The code segments of an actor correspond to the objects that implement the actor's function FUN. Two other model properties are inheritance and abstraction. To incorporate inheritance, the component $INHERITS(a)$ of an actor a is used to represent whether an actor inherits the functionality of one or more classes of actors (multiple inheritance), or is a primitive one (used in place of component $TYPE(a)$ of EDFG; component $\delta(a)$ is not yet used in IDFG). Abstraction can be used to improve reusability when the specified user interface will be implemented by the system.

Links are typed (mainly to distinguish between the components of a situation). The types of links currently used by SEPDS are contained in the set { *user action, system action, object condition, goal, incommunication, outcommunication* }. To improve expressibility, more link types can be added to this set. Links are used to describe different actor roles (e.g. "goal" links "assign" actors to user plans) and perspectives with respect to application functionality (e.g. the organizational perspective is represented by "action" and "communication" links).

State transitions take place only as a consequence of an event (that is, a user or system action), and *every actor must have a link of type "user action" or "system action" in its IFS*. In addition, each user action must belong to a goal–leading sequence, that is why, *each actor has also a link of type "goal" in its IFS and OFS* (a "goal" link in the OFS is used to signal successful goal achievement). Therefore, we explicitly represent all the actions (events) that may take place in the system, as well as all the goals that may be achieved using it. In addition, we can use several links of type "object condition" to represent the current state of the screen objects, thus accounting for the screen effects of user and system actions and conforming with the "principle of observability" [11].

To transparently support interaction across distributed contexts, as well as to model user–system communication, we use a special link type, the "communication" type. In effect, there exist "incommunication" and "outcommunication" link types to account for the direction of communication. Actors that contain rules that result in intercontextual communication are called "communication actors". On the other hand, links of type "system action" are used to model system–initiated communication among actors of the same graph.

4 Combining the Models

EDFG and IDFG have many common properties that make their combination rather straightforward: they use the same specification methaphor, while based on the same underlying mathematical model. This leads to a uniform treatment by the tools of SEPDS, and consequently facilitates the efficient prototyping and production of distributed interactive applications. In both models, inheritance and abstraction are incorporated using "templates", which are abstractions of the behavior of actors based on the description of their input and output links and firing pre- and postconditions. Templates may be archived and reused based on this description; they may also be combined to form abstractions of complex actor behavior (note that template combination is a different process than actor combination that is described in the following).

However, there also exist differences between the two models, that concern mostly their different semantic interpretation, and their distinct role in application specification. In EDFG, links represent data and consequently, link types represent data types, while actors represent application functions. In IDFG,

links represent events and link types represent different perspectives, while actors represent user interface functionality.

Thus, in a complete application graph, actors can be of two kinds: action actors and context actors. With each action offered by the application to its users (that is, with each command that is transferred by the user interface to the underlying application), an action actor is associated. The number of action actors is finite and equal to all the commands supported by the application. Such an actor has a simple behavioural part and fires when the user performs the appropriate action. In order for it to be ready–to–fire, all links in its subset of IFS specified by its PRE function except "user or system action" links must already contain tokens. This means that the user interface must have reached the appropriate state (as represented by the condition links) and the actor must belong in one of the contexts the user is currently working with (as represented by the "goal" link) for the user action to be available. Its functional part contains the code that implements the application command, and produces the appropriate effects (modelled with the production of tokens in the actor's subset of OFS specified by its $POST$ function). This code is represented with an EDFG subgraph.

To model context of operation and to support the goal–based structuring of user actions, context actors are used. These have a behavioural part that contains many rules, while their functional part will in the future be associated with some user interface widget to represent the action on the screen. Their functionality is to correctly interpret user actions in order to appropriately decompose user goals into subgoals, so that eventually the correct action actor will fire. To infer the context of operation, these actors contain rules that fire depending on the user interface action that the user performs. Context actors may be formed by combining action actors or context actors; this process may be applied an adequate number of times so as to represent all user goals and subgoals.

In this way, subgraphs that correspond to user goals can be defined, with the context of user actions encapsulated in their structure. As far as goals and user and system actions are concerned, the following rule holds: *lower level goals are derived from user or system actions and goals of the next higher level.* To achieve such a transformation when a context actor is formed, Primitive Graphs (PGs) [10] are used to specify the type of the context actor.

For example, suppose that one wants to specify interaction with a menu that has four items, the second of which opens to a submenu. Interaction with such a menu is depicted in figure 1. Note that a context actor (MENU) is used to represent the entire menu functionality, while action actors (C1, C3, C4) are used to represent the functions implemented by menu items 1, 3 and 4 respectively[1].

Menu item 2 is represented by another context actor (C2), since when selected, it opens to a second–level menu. Also note how action actors can be straightforwardly analyzed into EDFG actors (the semantics of this decomposition are not explicitly represented due to lack of space).

At any moment, there exists a number of actors (the actor–ready list) each of which contains tokens in the subset of its IFS links specified by its PRE function, except the "action" link. These actors represent the actions that are available to the user. Traditional DFG models interpret the notion of state as the distribution of tokens on the graph links. Our model extends this notion by defining a state as the set of actors in the actor–ready list, or equivalently, the set of user or system actions that the actors in the actor–ready list represent. Since these actions correspond to goals in a lower–level, it may be equivalently

[1] in this figure thick and plain circles represent user action and other links, respectively, thick and plain rectangles represent action and context actors, respectively, and dotted rectangles represent EDFG actors

stated that a state is represented with the set of goals that may be achieved as a consequence of user or system actions permitted by the actors in the actor–ready list. State transitions occur as a consequence of an actor firing which causes the output of tokens in the subset of actor's OFS links specified by its $POST$ function, thus modifying the actor–ready list.

Since, however, an actor firing depends on its IFS, it is clear that among a set of otherwise identical actors that are ready to fire (the actor–ready list), the one that fires is determined by the link of type "user or system action". This property can be used for the resolution of firing conflicts: all actors that can eventually fire, do so, and the consequences of firing appear in the graph in the form of a new marking.

Referring to the example of figure 1 once more, PG EN creates one token on every "goal" link of the constituent actors when user moves the mouse over the menu. Thus, actors C1 to C4 become ready–to–fire. The one that will eventually fire is determined by the next user action (clicking on one of the menu items), that will create one token on the "user action" link of one of these actors. This actor is subsequently analyzed in the same way. Note that menu closes after one item is selected, as specified by PG OPG, which produces one output token when a token is created on one of its input links.

In an interactive application, there exist several "loci of control", representing the potential for the next user action; these loci are represented in SEPDS by the actor–ready list. Furthermore, the existence of an "external event–handler" (EEH) is assumed, which gets user input and sends it to the IDFGs. EEH does not wait for response to the token it communicates; instead, *it communicates a token each time a user action is recorded and identified.* The task of EEH is not only to capture each user action, but also to assign to it the proper semantics depending on the context. This mechanism, however, is invisible to the user.

5 Conclusions

In this paper a solution to the problem of combining an interaction model with an application data model was presented. This solution, which has been adopted in SEPDS, consists of using EDFG for application functionality specification, and IDFG for specification of interaction. These two models are both based on Data Flow Graphs and can be combined straightforwardly.

Designers that use SEPDS may either describe an application in the traditional bottom–up way (by first describing the functionality of the application to be produced using EDFG and then by specifying the interactive features of the application using IDFG) or in a top–down way (first, the interactive features are specified, then the user goals are decomposed and the action paths are defined, and finally, the application functionality is described). In both methods EDFG graphs must be associated with the action actors of IDFG and IDFG actors must be combined to represent user interface functions. Note, however, that in the latter method the applications that will be produced are user–centered in that they directly incorporate the user perspective. Then the prototyping tools of SEPDS may be used to test the appearance and functionality of the application.

The importance of this process lies in that designers do not have to construct the user interface, as is usually the case with other user interface generators. Instead, they specify the flow of interaction between projected end–users and the application to be produced. To do this, designers must specify the actions that will at any moment be available to end–users, the effects that these actions will produce on the screen, as well as the goals that may be achieved using the application. That is why we claim that the presented model enables designers to think in users terms.

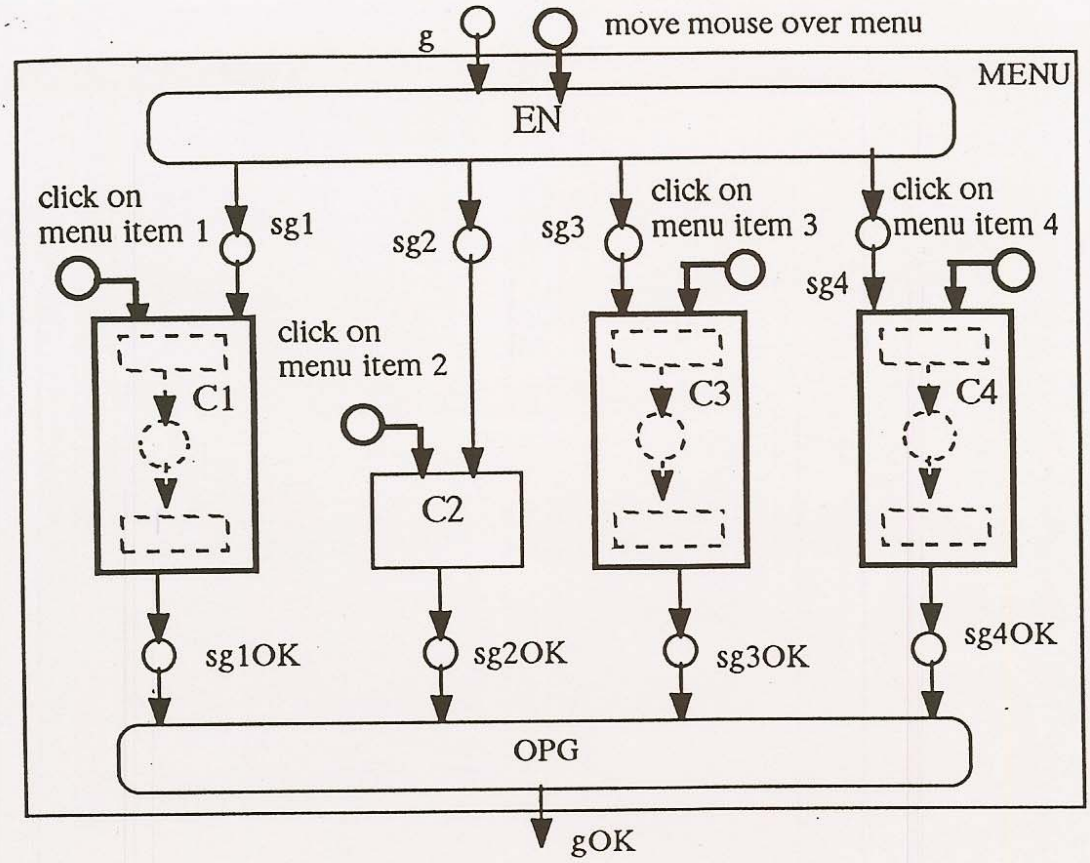

Figure 1: An example of the usage of EDFG and IDFG models

A prototype of SEPDS has been recently completed, but it does not yet support the IDFG model (the model, however, has been independently applied to the descritpion of several interactive systems). We are currently in the process of defining a user interface construction methodology that together with the integration of IDFG, will enable us to test the feasibility of a semi–automated production of user interfaces.

Acknowledgements
The authors wish to thank Mr. P. Fitsilis for fruitful discussions on the usability of the approach, and the anonymous reviewers for their constructive comments.

References

[1] J.D. Foley, D. J. M. J. de Baar and K.E. Mullet, *Coupling application design and user interface design.* Proceedings of the CHI92 Conference: Striking a balance, May 3-7, 1992, Monterey, USA, pp 259-266.

[2] J. Bonar and B. Liffick, *Communicating with high–level plans.* In Intelligent User Interfaces (J. Sullivan and S. Tyler eds), ACM Press, 1991, pp 129-157.

[3] R. Buhr, G. Karam, C. Hayes and C. Woodside, *Software CAD: A revolutionary approach.* IEEE Trans. Softw. Eng., SE-15(3), March 1989.

[4] B. Curtis, M.I. Kellner and J. Over, *Process modeling.* Comm. of the ACM, 35(9), September 1992, pp 75-90.

[5] A. J. Dix and C. Runciman, *Abstract models of interactive systems.* In Proceedings of the British Computer Society Conference on People and Computers: Designing the Interface (P. Johnson and S. Cook eds), Cambridge University Press, 1985, pp 13-22.

[6] P. Henderson, editor, *Proceedings of the second SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments.* ACM SIGPLAN Notices, vol 22, January 1989.

[7] C. Jard, J. Monin and R. Groz, *Development of VEDA, a prototyping tool for distributed algorithms.* IEEE Trans. Softw. Eng., SE-14(3), March 1988.

[8] A. Kameas, S. Papadimitriou, P. Pintelas and G. Pavlides, *IDFG: an interactive applications specification model with phenomenological properties.* Proceedings of the EUROMICRO93 Conference, September 6-9, 1993, Barcelona, Spain.

[9] A. Levy, J. van Katwijk, G. Pavlides and F. Tolsma, *SEPDS: A support environment for prototyping distributed systems.* Proceedings of the 1st International Conference on System Integration, April 1990, New Jersey, USA.

[10] S. Papadimitriou, A. Kameas, P. Fitsilis and G. Pavlides, *A new compression technique for tools that use data–flow graphs to model distributed real–time applications.* Proceedings of the 5th International Conference on Software Engineering and its Applications, December 7-11 1992, Toulouse, France, pp 235-244.

[11] H. Thimbleby, *User Interface Design.* ACM Press, 1990, p 470.