Clover Architecture for Groupware

Yann Laurillau
Laboratoire CLIPS/IMAG
University of Grenoble
Domaine Universitaire, BP 53
38041 Grenoble, FRANCE
Yann.Laurillau@imag.fr

Laurence Nigay[1]
University of Glasgow
Department of Computer Science
17 Lilybank Gardens
Glasgow G12 8QQ
laurence@dcs.gla.ac.uk

ABSTRACT
In this paper we present the Clover architectural model, a new conceptual architectural model for groupware. Our model results from the combination of the layer approach of Dewan's generic architecture with the functional decomposition of the Clover design model. The Clover design model defines three classes of services that a groupware application may support, namely, production, communication and coordination services. The three classes of services can be found in each functional layer of our model. Our model is illustrated with a working system, the CoVitesse system, its software being organized according to our Clover architectural model.

Keywords
Conceptual Software Architecture, Clover Design Model.

INTRODUCTION
People's interest in collaborating with others is growing daily and in particular on the internet. This is not only in order to read web pages or download files but also to communicate, to play multi-user games or to exchange data such as music files. In a nutshell, to interact with others. This leads to an outbreak of a multitude of multi-user systems (or groupware) designed to chat, such as ICQ, to play, such as Quake or to share mp3 files, such as the well-known Napster (peer-to-peer). Despite the multiplicity of existing groupware, their development still remains complex and unsystematic. It is widely recognized that, although the adhoc development of software is acceptable for throw-away prototypes, architectural design of complex systems can no longer simply emerge from craft skills. In this paper we address the architectural design of groupware by defining a conceptual software architectural model for groupware.

A conceptual software architecture is an organization of computational elements and the description of their interactions. A conceptual software architecture model defines a vocabulary of design elements, imposes configuration constraints on these elements, determines a semantic interpretation that gives meaning to the system description, and enables analysis of the system properties [22].

For example, Dewan's generic architectural model [8] structures a groupware application as a stack of shared and replicated layers which communicate with each other by exchanging events. A conceptual software architecture is then mapped into an implementation architecture dependent on the software tools available. Software tools for the construction of groupware such as the Groupkit interaction toolkit [21] or the COCA [16] and DragonFly [1] platforms will not eliminate conceptual architectural issues as long as the construction of these systems requires programming. Clearly, a conceptual model for identifying and organizing the components of groupware is still a necessity. As explained in [5], without an adequate architectural framework, the construction of groupware and in general interactive systems is hard to achieve, the resulting software is difficult to maintain and iterative refinement is impossible.

Several conceptual software architectural models for groupware exist, such as Dewan's generic architecture [8], Clock [11] and PAC* [5]. Our Clover architectural model is complementary to the existing models by defining at a finer grain how functionalities should be structured at each level of abstraction in the architecture. To do so, we base our architectural model on the Clover design model [9][23]. The Clover design model defines three classes of services: production, communication and coordination services. During the system design phase, the three types of services must be identified and their access harmoniously combined in the user interface. Our Clover architectural model makes such system design concepts explicit within the software. Indeed, because software architecture modeling is a design activity at the interface between the system design field and the programming field, software architecture must take into account attributes of both fields: these attributes include properties of the designed system and of the future software to be implemented [17].

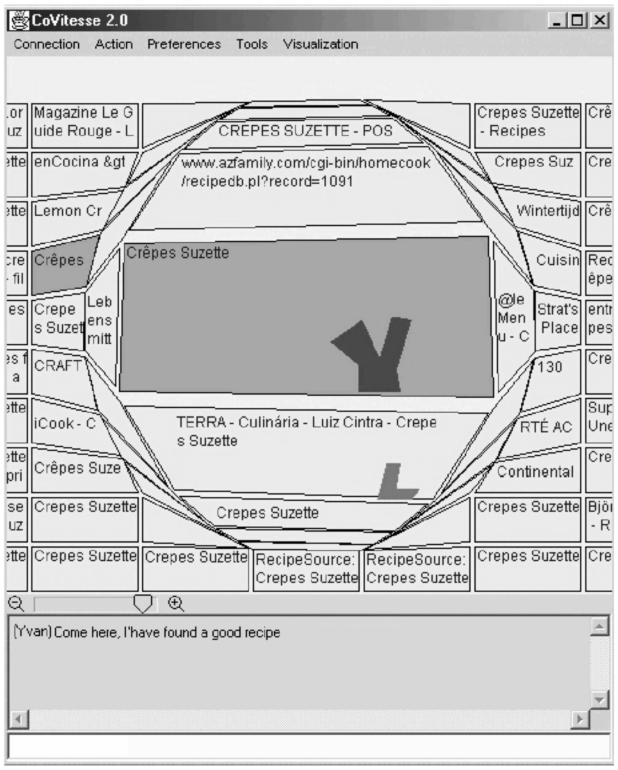
The structure of the paper is as follows: first, we introduce the Clover design model and stress its impact on the design of the system as well as on the design of the user interface. We then present the main characteristics of three architectural models, Arch, Dewan's architecture and PAC*, all three of which our Clover metamodel relies on. Our Clover architectural model is a metamodel i.e. several Clover architectural models can be derived from it. We first present one Clover architectural model and then the generalization, the metamodel. We close with the presentation of the clover architecture of a running system, CoVitesse, which enables collaborative navigation on the World Wide Web. The discussion will be illustrated with

1. On sabbatical from the University of Grenoble, CLIPS Laboratory, BP 53, 38041 Grenoble Cedex 9, France.

two systems, CoVitesse system, whose main features are presented in the next section.

AN ILLUSTRATIVE EXAMPLE: COVITESSE SYSTEM

We applied our Clover architectural model to the software design of the CoVitesse system.



(Yvan) Come here, l'have found a good recipe

Figure 1: Snapshot from the CoVitesse system.

CoVitesse [7][14] is a groupware application that enables collaborative navigation on the WWW. Users sharing the same information space can cooperate to seek information. This system is built on the Vitesse system [19], a single-user application that visualizes the results of a query submitted to a search engine on the WWW. As shown in Figure 1, CoVitesse displays the overall set of results: each retrieved page (node) is displayed by a polygon. Users are represented by colored shapes (a red Y for Yann and a blue L for Laurence). When a user wants to visit a page, she/he double-clicks on the corresponding polygon and the web page is downloaded in a separate browser. Darker polygons indicate visited web pages. At any time, a user can see all other users moving in the space, use the chat box (which is below the information space) to communicate with other users, or organize her/his own caddy which contains the marked pages.

CoVitesse allows users to create groups. A group is identified by a name and a color. For example, in Figure 1, Yann, represented by the shape Y, is a member of a group identified with a red color. In the current version of the system, four kinds of groups, i.e. collaborative navigational tasks, are available: guided tour, opportunistic navigation, shared navigation and cooperative navigation. According to the group type, different functionalities are available. For example, within a group defined as an opportunistic navigation group, a member can take control of the navigation of the other members; such a functionality is not possible with a Shared navigation group. Moreover, access rights to data are different according to group types; rights are imposed on the group caddy (i.e. the pages selected by the group) as well as on the group preferences. Group preferences include the information related to the group, the choice or not to publish the gathered results and the publication filter applied on the results. For example, any member of an opportunistic navigation can modify the group preferences. More details about these navigation types can be found in [14].

CoVitesse provides persistent access to all the data, modified or produced during a session. These data include information about users and groups such as the avatar shape, the gathered results and the preferences. These data are protected with a simple mechanism of username/password. Then, when a user starts a new session, she/he recovers her/his own private data.

CLOVER DESIGN MODEL

Description

The Clover design model [9][23] provides a high level partitioning for reasoning about the collaborative services a groupware application may support. As shown in Figure 2, a groupware application covers three kinds of services: production, communication and coordination.

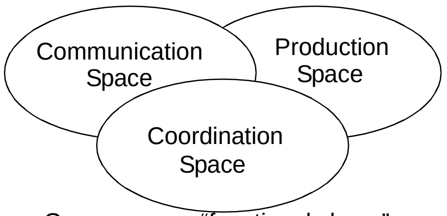


Figure 2: Groupware as “functional clover”.

- The production space refers to the objects produced by a group activity or to the objects shared by multiple users. In the CoVitesse system, the production is the set of results gathered by users and groups during a session.
- The communication space refers to person-to-person communication such as e-mail, relay chat, mediaspace. The CoVitesse system provides a chat room for users navigating in the same information space.
- The coordination space covers activities dependencies including temporal relationships between the multi-user activities. It also refers to the relationships between actors and activities. In the CoVitesse system, coordination tasks include creating, joining, leaving a group or accepting a new member. Looking at the avatars moving within the information space is also a coordination task.

However, not all the existing groupware support the three facets of the Clover design model. This point is further discussed in the following section.

CSCW Design Issues

The Clover design model is useful for the design of groupware during the functional specification phase because it defines three classes of functionalities that a groupware application may support. For example, the

CoVitesse system supports the following three types of functionalities:

- *Production*: the system manages a caddy of collected results for each user and group. It is possible to modify the group caddy according to the defined access rights.
- *Communication*: the system provides a simple chat box. Currently, the communication group is composed of all the users.
- *Coordination*: the system provides services to join/leave a session, to create/join/leave a working group. In addition, coordination is done when a user moves her/his avatar in the information space or when s/he modifies the preferences.

As we mentioned in the previous section, not all the existing groupware support the three types of functionalities. For example, a mediaspace, a system dedicated to informal communication to enhance team awareness, is built upon communication functionalities and few coordination functionalities such as joining or leaving a session. In this example, there is no support for production. In addition some approaches treat the coordination functionalities as special in contrast to the production and communication functionalities: coordination functionalities are separated from the system itself. For example in [8] the session manager dedicated to coordination is modeled as an external software component. Another example is the COCA platform [16]. The platform is dedicated to coordination and is able to manage tools including conference tools or whiteboards based on a set of rules defined in a prolog-like language. These rules dedicated to coordination are interpreted at the runtime by the COCA virtual machine. The key idea of this platform is to extract the coordination functionalities from the system. The platform therefore enables the reuse of non-collaborative systems, but it provides no support for production and communication. Moreover an existing groupware application may include its own coordination model which may not match with the COCA one.

When the Clover model defines three types of functionalities, it also defines three classes of data structures manipulated by these functionalities. For example let us consider the implementation of the notion of group in a chat box used to manage a lecture. We consider two types of actors: the lecturer and the audience. The system implements coordination functions, to join a session and to assign roles (i.e., lecturer or audience) and enables communication between the lecturer and the audience. In order to avoid interference generated by the audience chatting, we should consider two communication channels: one channel for communication between the lecturer and the audience and another channel between the members of the audience excluding the lecturer. The system therefore maintains two data structures related to the notion of group:

- one structure containing the representation of a lecturer and an audience, for coordination and communication,
- one structure containing the representation of an audience with its members and excluding the lecturer, dedicated to communication only.

The example illustrates the fact that data structures can be classified according to the three Clover facets. In addition the example shows that the same data structure can be shared and manipulated by different Clover functionalities. The data structure therefore belongs to one of the intersections of the three facets in Figure 2. To further illustrate this last point we consider CoVitesse: when a user moves her/his avatar in the information space:

- the shape is drawn at the new coordinates in each user's view: a coordination functionality;
- the URL is added to the group history: a production functionality.

One data structure is used for modeling the position of an avatar in the information space and several Clover functionalities manipulate this unique structure.

In conclusion, the Clover design model can be used as a guideline in the design of groupware for specifying the functionalities as well as the data structures. The usefulness of the model for the design of the user interface is more problematic.

HCI Design Issues

While the Clover model is useful for identifying the services provided by groupware, it should not be used as a guideline for designing the user interface. For example, having separate windows or panels dedicated to each Clover space will lead to a complex interaction. The first design of the CoVitesse user interface was based on the Clover design model. The following three windows are part of the first CoVitesse user interface:

- a window dedicated to the production containing the information space and the marked web pages,
- a window dedicated to communication that includes textual chat, and,
- a third window dedicated to coordination that allows the user to create and edit a group.

Informal tests with groups of four users having to perform a scenario (i.e., collect a list of the ten most relevant web sites about human-computer interaction), lead us to conclude that the information space and the chat box should be combined at the user interface (combination of production and communication). Indeed, at the end of the test sessions, users complained about a lack of awareness: they were not able to perform a navigational task and to look at chat exchanges at the same time.

In addition to our own experience with CoVitesse, several groupware systems combine the services provided by each of the three Clover spaces, making them accessible and observable through appropriate user interface composite objects. For example, as shown in Figure 3, the collaborative scrollbar introduced in [5] and based on the multi-user scrollbar of the SASSE editor [2] covers functionalities of the three spaces. The right scrollbar, dedicated to production, is an ordinary scrollbar for scrolling the text. The left scrollbar, dedicated to communication and coordination, contains multiple elevators, one per remote user who can be seen in a video thumbnail inside the elevator. On the one hand, from a functionality viewpoint, it is easy to observe that the coordination is covered by the current position of remote users in the text and communication by a video service. On the other hand, from a user interface viewpoint, a single interaction object (a scrollbar) is used for coordination as well as communication.

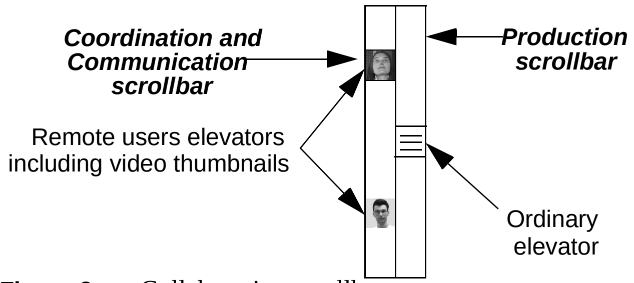

Coordination and Communication scrollbar
Production scrollbar
Remote users elevators including video thumbnails
Ordinary elevator

Figure 3: Collaborative scrollbar.

In conclusion, our own experience as well as existing groupware widgets underline the fact that the Clover design model is useful in identifying the funtionalities that the user interface should provide but should not be applied as a guideline for designing the user interface itself. Such conclusions have a direct impact on our Clover architectural model. Before presenting our model, we show the main characteristics of three architectural models on which our model relies.

SOFTWARE ARCHITECTURES

This section contains the description of three architectural models, Arch, Dewan's generic architecture and PAC*. The three models serve as foundations for our Clover architectural metamodel.

Arch model

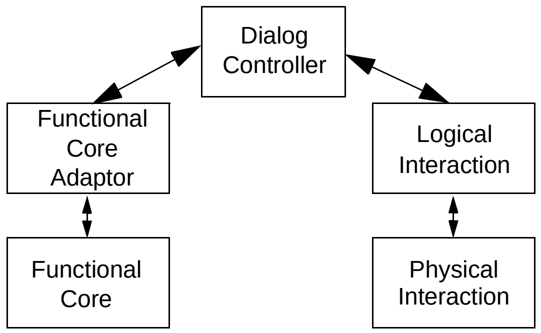


Figure 4: Arch model.

Software architectures for groupware are often based on architectural models for single-user interactive systems. In this field, the Arch model [3], an extension of the Seeheim model [10], is a reference model that defines a canonical functional decomposition of an interactive system. Arch advocates domain dependent components as well as user interface components. As shown in Figure 4, the Arch model breaks up an interactive system into five layers or components:

- The Functional Core component implements domain-specific concepts and functions, in a presentation independent way. The data structures managed by this component are domain objects.
- The Functional Core Adapter component serves as a mediator between the Dialog Controller and the Functional Core. Data exchanged with the Functional Core are the domain objects that the Functional Core exports to the user. Data exchanged with the Dialog Controller are conceptual objects. These define perspectives on domain objects intended to match the user's mental representation of domain concepts.
- The Physical Interaction component denotes the underlying platform, both software and hardware (interaction devices). It supports the physical interaction with the user and corresponds to the services of a User Interface toolkit.
- The Logical Interaction component implements the perceivable behavior of the application for outputs as well as inputs. It relies on the Physical Interaction component. The Logical Interaction component is usually compared to an abstract User Interface toolkit. It serves as an adapter between the Physical Interaction component and the Dialog Controller.
- The Dialog Controller component is the keystone of the arch. It has the responsibility for task-level sequencing. It manages both conceptual objects and presentation objects.

Dewan's Generic Architecture for Groupware

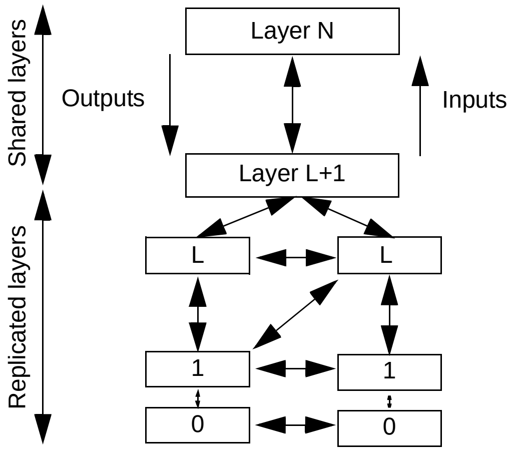


Figure 5: Dewan's generic collaborative architecture.

Dewan's generic collaborative architecture [8] can be seen as a generalization of the "zipper model" [20] and the Arch model. Dewan's architecture structures a groupware application into a variable number of layers from the domain-dependent level to the hardware level. A layer is a software component corresponding to a specific level of abstraction. As shown in Figure 5, the top-most layer corresponds to the semantic layer while the bottom-most layer corresponds to the hardware layer. To make a parallel with the Arch model, the top-most layer coincides with the Arch Functional Core component and the bottom-most layer coincides with the Arch Physical Interaction component.

The overall structure is composed of a stem and several branches. The stem, as shown in Figure 5, is composed of shared layers (layers L+1 to N). A branch is composed of replicated layers (layers 0 to L). The layer L of Figure 5 defines the branch point of replicated or versioned layers [8]. A branch contains private layers for each user of the application. Moreover the objects managed by layers of a branch are all private objects of a user. Conversely, shared layers and shared objects are public. The layers communicate with each other using two types of events: interaction and collaboration events. Interaction events denote events sent up (input events) and down (output events) between layers. Events sent between layers of different branches are collaboration events.

In the rest of the discussion, we shall often use the terms layer and component interchangeably. Although a software component can encompass several layers, for simplicity we assume that there is one layer per component.

PAC* model

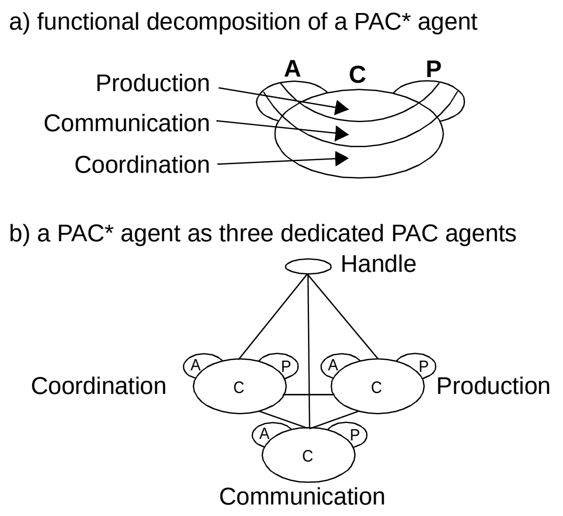

a) functional decomposition of a PAC* agent
A
C
P
Production
Communication
Coordination
b) a PAC* agent as three dedicated PAC agents
Handle
Coordination
Production
Communication
A
P
C

Figure 6: PAC* architectural model.

PAC* [5] is the collaborative extension of the PAC-Amodeus hybrid architectural model [18]. PAC-Amodeus reuses the main components of the Arch model and populates the Dialog Controller with PAC agents. A PAC agent is composed of three facets:

- a Presentation facet which defines the user interface of the agent,
- an Abstraction facet which manages the domain concepts, and,
- a Control facet which manages the links and constraints between its two surrounding facets (i.e., the Presentation and the Abstraction facets) as well as its relationships with other agents.

A PAC* agent, as shown in Figure 6 (a), is a PAC agent in which each facet is structured along with the three functional classes of the groupware clover. Alternatively, a PAC* agent can be seen as a cluster of three PAC agents, each agent dedicated to one functional class of the groupware clover. The cluster of PAC agents represented in Figure 6 (b) is a hybrid form of a PAC* agent: the three clover agents can communicate with each others via their Control facets and a single handle is in charge of communication with the external world (i.e., other agents).

CLOVER ARCHITECTURE

The Clover architectural metamodel relies on the three above models. From our metamodel, several Clover architectural models can be derived. In this section we present one Clover architectural model derived from the metamodel, the latter being presented in the next section.

Description

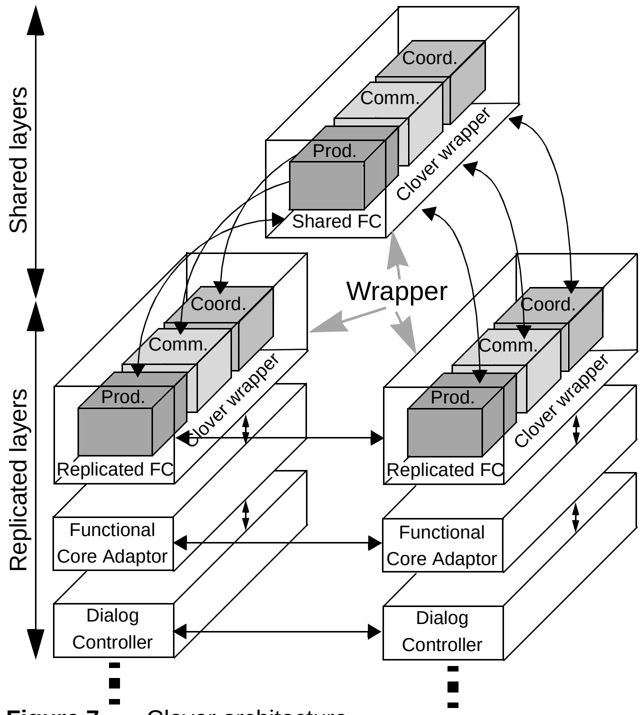


Figure 7: Clover architecture.

The Clover architectural model of Figure 7 is derived from Dewan's model using the functional layers of the Arch model [3]. Nevertheless instead of the five layers of the Arch model, our model advocates six layers: the Arch Functional Core (FC) component is split into two components: the shared and replicated FCs. In Figure 7, to avoid a visual overload, we have not drawn the Arch Logical and Physical Interaction components, which are located below the Dialog Controller in the stack. In addition, some arrows, between layers of the two branches are not drawn.

Having defined the canonical component decomposition, let us now consider the global architecture made of shared and replicated components. All the Arch components are replicated components except the shared FC. In this model, we focus on the functional components and we have made no assumptions about the Dialog Controller and the Presentation Components. Compared to the PAC* model, in which the FC is a completely shared layer, the FC is split here in two layers: one is replicated and private; the other is shared and public. The latter provides shared domain-dependent objects and functions that are manipulated by all the users during the interaction. Conversely, the replicated FC maintains the state of a single user and manages private domain-dependent objects. As in Dewan’s model, the collection of shared and user’s private objects defines the interaction state for a particular user. This point is illustrated with the example of the software design of CoVitesse in the last section.

The originality of this architecture comes from the Clover decomposition of the Shared and Replicated FCs. The FCs are made of a wrapper which embeds three sub-FCs, each sub-FC being dedicated to one of three functional classes of the groupware clover. Each Clover FCs manipulates semantic objects dedicated to one of the Clover functionalities and performs specific processing functions on their objects. The wrapper approach is similar to the software component oriented programming in CORBA. The design rationale for the wrapper is threefold:

- The wrapper encapsulates the three Clover FCs and acts as a functional glue. The wrapper is also responsible for communication with other layers. Because the wrapper is a software interface that hides the Clover partitioning, it enables communication with non-clover aware layers. For example the shared FC wrapper serves as a mediator between the three Clover shared FCs and the Functional Core Adapter. The wrapper allows us to stack a Clover partitioned component with a component that is not collaboration aware or Clover aware.
- The wrapper maintains a common state, i.e. the common semantic objects, among the production, communication and coordination FCs. Such objects common to the three Clover spaces have been identified and illustrated in the section entitled "Clover design model".
- The wrapper provides functionalities including the system services and single-user functionalities, which are not intrinsically collaboration aware or Clover aware.

The communication between layers in the stack and between peer layers follows the same rules as described in Dewan’s model. Moreover interaction and collaboration events are categorized into three classes of events: production, communication and coordination events. This enables direct communication between Clover FCs. Nevertheless the model maintains a general event for communication between layers that are not Clover aware.

Finally, we have explained in a previous section that the Clover model should not be used as a guideline for designing the user interface. As a consequence, we only partitioned the FC component according to the three Clover functional classes and not the presentation components (Logical and Physical Interaction components). Furthermore it is important to note that the model is one example of a model derived from our metamodel: in this model we have unzipped the architecture at the FC layer, the branching point being the Replicated FC. Different branching points are possible. Fixing the branching point is a design issue and defines the versioning/replication architectural degree, as defined in [8]. In our Clover architecture, the Replicated FC being the branching point, the corresponding replication degree is high. The Clover model therefore allows relaxed WISIWYS (What I See Is What You See). Indeed a low replication (i.e. an architecture unzipped at the Physical or Logical Interaction layers) implies strict WISIWYS.

An Extension of PAC*

Our Clover architectural model is complementary to the PAC* model. Indeed in the PAC* model, a PAC* agent of the Dialog Controller exchanges events with the Functional Core via the Functional Core adapter. In our Clover model, if the Dialog Controller is populated by PAC* agents (Figure 6), each Clover Abstraction facet of a PAC* agent will communicate with the corresponding Clover Replicated FCs. But for a high modifiability of the code, the communication will not be direct but will be via the Functional Core Adapter that is not necessarily Clover aware, and then via the Replicated FC wrapper.

Moreover in Figure 6(b), a PAC* agent contains a handle. This latter is responsible for managing the communication with the external world, i.e. the other agents. The role of the handle is one of the roles fulfilled by a wrapper in our Clover model. The wrapper manages communication between the three Clover FCs and their surrounding layers.

CLOVER METAMODEL

Having presented one Clover architectural model, we now describe the generalization: the Clover metamodel.

Description

The Clover architectural metamodel of Figure 8 structures a groupware application into a variable number of layers, as opposed to the previously presented Clover model which is based on the five layers of the Arch model. Compared with the previous Clover model of Figure 7, Layer L corresponds to the replicated FC and Layer L+1 to the Shared FC.

The originality of this metamodel comes from the Clover decomposition of the software units that compose the branches and the stem. As shown in Figure 8, each unit of the model contains a wrapper that encapsulates three Clover sub-components: production, communication and coordination sub-components. Interaction and collaboration events exchanged between layers are subsequently categorized into three classes of events: production, communication and coordination events. The metamodel nevertheless authorizes a general event for communication between layers that are not Clover aware. In Figure 8, in order to avoid a visual overload, we have not drawn every arrow corresponding to an exchange of events between layers.

As explained in the previous section, one role of a layer wrapper, which encompasses a Clover’s structure, is to hide the Clover breakdown from the other layers. Communication between a Clover aware layer and a non-Clover aware layer is thus possible. Indeed, a layer does not necessarily implement a collaboration semantic and is therefore not necessarily decomposed according to the three Clover functionalities. This is usually the case for the lowest layers, which are dedicated to the physical and logical interaction with a user. For example, as shown in Figure 8, Layer 0 (the leaf of the branch) which

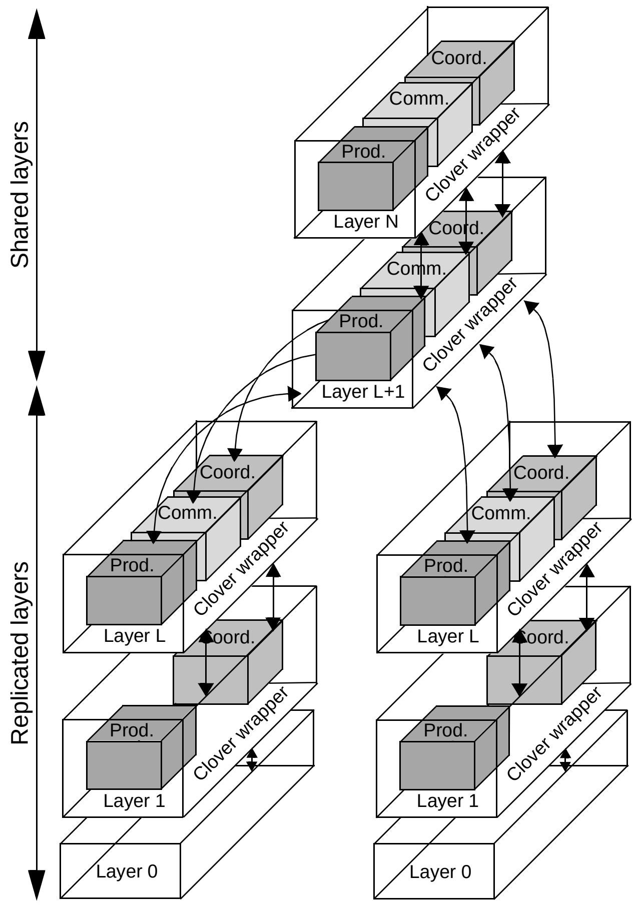


Figure 8: Generalized Clover architecture.

corresponds to the hardware layer, is typically a non-Clover aware layer.

Moreover the full decomposition of a layer according to the three Clover functionalities is not mandatory. A layer may include Production and Coordination functionalities and no Communication functionality: this is for example the case for Layer 1 in Figure 8.

BENEFITS AND PROPERTIES

The Clover functional partitioning establishes a direct mapping between the design concepts and the software architecture modeling. This partitioning provides a clear guide in the organization of the functionalities identified during the design phase. In addition it is complementary to the traditional partitioning of functionalities into components, where each of them corresponds to one level of abstraction as for instance in Dewan's architecture or Arch model. Indeed, for a component corresponding to one level of abstraction, the Clover metamodel advocates three sub-components dedicated to production, communication and coordination. Such a highly modular implementation satisfies the modifiability, reusability and extensibility properties. These properties are crucial for supporting a user-centered design. For example, in the CoVitesse system, it would be possible to add a video service as a new communication channel by modifying the Shared and Replicated Functional Core dedicated to communication without endangering the other components. Moreover modularity is a well-known software engineering mechanism for reducing the cost of development. Based on our own experience, applying the Clover architectural model to the CoVitesse software design made it easy to develop parts of the system without having a global view of the overall code. For example, the three Shared Functional Cores and the three Replicated Functional Cores, described in the following paragraph, have been developed and tested independently.

The metamodel does not fix the number of layers: layers can be used to increase the separation of functionalities i.e. to increase the modularity of the code. This approach is orthogonal to the Clover breakdown of a layer.

The Clover breakdown of a layer can be partial. The metamodel therefore allows a pile of several layers with different Clover breakdowns. For example, instant messaging (communication functionality) is usually used for managing coordination among users [24]. To implement such a coordination which is based on communication, a shared layer dedicated to coordination is stacked on top of a shared layer that contains communication functionalities only.

In our metamodel the coordination, production and communication functionalities are all treated the same way. This is in contrast to previous studies [8][16] where the coordination functionalities are treated as special whereas the production and communication functionalities are not. For example, in Dewan's architecture [8], the session manager is a special component that should be located outside the architecture. This external component manages the users and the groups during a session. This component is therefore responsible for creating the tree and for connecting the branches with the stem. In the Clover metamodel, the session manager is a coordination component located in a stem layer. Indeed we argue that there is always a shared layer in the conceptual software architecture. The conceptual architecture is then mapped into an implementation architecture by assigning processes to components; the processes in turn must then be assigned to hosts. Different approaches to distribution can be applied to a conceptual architecture. Distribution and layer decomposition (or software component decomposition) are two orthogonal mechanisms. In particular, a shared layer at the conceptual level does not imply a central site at the implementation level.

Finally the Clover metamodel extends Dewan's generic architectural model [8]. An architecture based on Dewan's generic model is described by an awareness degree. The awareness degree is equal to the level of the highest layer that is collaboration-aware. Extending the awareness degree, we define a production-aware degree, a communication-aware and a coordination-aware degree. Such extensions will allow a more precise classification of existing systems from an architectural point of view, than the one presented in Table 9.1 in [8].

CLOVER MODEL IN PRACTICE: COVITESSE SOFTWARE DESIGN

In this section, we show how the Clover model can be used in practice by presenting the Clover software design solution of CoVitesse.

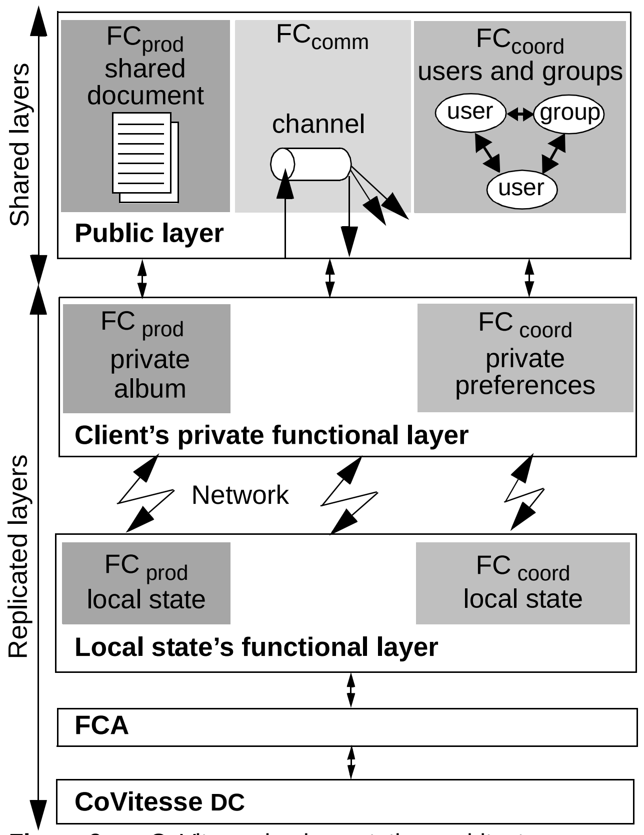


Figure 9: CoVitesse implementation architecture.

Figure 9 illustrates the application of the Clover model to the software design of CoVitesse. At the conceptual level, we applied the Clover model of Figure 7 that includes a replicated Functional Core and a shared Functional Core. At the implementation level, CoVitesse is a client/server application written in Java. The shared components are on the server side and the replicated components on both sides (client and server). The communication between clients and the server is based on network streams.

We describe the CoVitesse architecture from the highest layer (top of Figure 9) to the lowest. The highest layer, the Shared Functional Core (FC), is made of a FCprod, a FCcomm, a FCcoord, and a meta-server. The meta-server is a wrapper: it encompasses the three Clover FCs and manages the access to the underlying network.

- *Shared Production Functional Core (FCprod)*:
  This component manages concepts related to production, represented as a document icon in FCprod of Figure 9. These concepts are document, reference, album and catalogue. A document is an elementary unit that is manipulated by a group. A reference is a pointer on a document. An album is a container, created and managed by authors: an album contains documents and references. A catalogue is a set of references. These four concepts are implemented as Java interfaces. There are several functions available including: the creation or obtaining of a catalogue, creation of an album, addition or removal of an author from an album, addition of a document in an album, modification of the content of an album. In CoVitesse, a reference is an URL, a document is the content of a web page, i.e. an HTML document, a catalogue is a set of URLs resulting from a query submitted to a search engine and an album is a set of URLs created by authors.

- *Shared Communication Functional Core (FCcomm)*:
  This component manages communication channels that are represented by a tube in FCcomm of Figure 9 The communication channel is the only concept manipulated by FCcomm. A client can subscribe or unsubscribe to a communication channel and can post a message on the channel. A channel is implemented in Java. In CoVitesse, the unique communication channel is a chat channel, which can be compared to a mailing list: when a user posts a message on this channel, the message is broadcast to all the subscribers.

- *Shared Coordination Functional Core (FCcoord)*:
  This component has the responsibility for coordinating users and groups. It first hosts a database of users and groups. Therefore this component allows persistent access to the description of the users and the groups. Moreover, this component provides mechanisms for concurrency control and for consistency control. For instance, it synchronizes a concurrent access for two members to modify group preferences. This component also notifies all the clients, for consistency purposes, when a user modifies her/his own private preferences. Consequently the concepts manipulated by FCcoord are related to users and groups; these concepts are implemented as Java interfaces.

  - *User*: Two functions are implemented to read and modify information about a user as well as two functions to read and modify the access rights to those pieces of information. This implementation allows the developer to define the user's description according to the nature of tha groupware application. In CoVitesse, a user is described by a name, a password, a geometrical shape, filtering preferences, an E-mail address, etc. The user is identified by her/his username and password.

  - *Group*: A Group is defined by, a set of members, rights for joining a group and a minimal set of functions. These functions include setting or obtaining information about a group, creating a group, joining a group (a user sends a request to all the members who accept or do not accept the newcomer), leaving a group. In CoVitesse, a group is described by: a group name, a color, filtering preferences, a navigation type and navigation preferences.

As shown in Figure 9, the layer under the Shared Functional Core is the Replicated Functional Core. At the implementation level, the Replicated Functional Core is split into two parts: one part on the server side and one part on the client side. The part on the client side maintains a duplicated version of the current session, which is originally saved on the server side. This mechanism of caching data is used to increase performance at runtime. The Shared Functional Core (on the server side as well as on the client side) is made of two Clover FCs, namely FCprod and FCcoord. These two Clover components maintain private data owned by the client. This design solution enables the server part to provide persistent access to those private data.

- Replicated FCcoord maintains the user's private preferences and communicates with other clients through the Shared Functional Core. Both replicated and shared components dedicated to coordination manipulate the same data about a user. Nevertheless only the replicated component is allowed to modify private data such as the username and password, while the shared component can read the private data about a user when it is published.

- Replicated FCprod hosts the user's private album containing the collected URLs. Moreover FCprod has access to the group album managed and protected by the Shared Functional Core dedicated to production.

In the current implementation of CoVitesse, there is no Replicated Functional Core dedicated to communication. Consequently the wrapper transfers communication events directly, in a bidirectional way, between the shared FCcomm and the Functional Core Adapter (FCA in Figure 9). However, future developments are planned for implementing this missing facet. For example, private aliases of recipients such as mailing-lists can be managed by the shared FCcomm. Moreover, we have planned to extend the shared FCcomm in order to allow private channels for chatting within a defined group of users.

In the rest of the architecture, the components do not implement a Clover structure. These components are the Functional Core Adapter (FCA), the Dialog Controller (DC) and the Logical and Physical Interaction components. In Figure 9, the Logical and Physical Interaction components are not drawn. They should be drawn under the CoVitesse DC.

VALIDATION OF THE CLOVER METAMODEL

Software tools for the construction of groupware will not eliminate architectural issues as long as the construction of groupware requires programming. Developers and maintainers of groupware need to rely on architectural models:

- for identifying software components,
- for organizing their interconnections,
- for reasoning about components and interconnections,
- for modifying and maintaining them in a productive way,
- for verifying ergonomic and software properties.

Nevertheless an architectural design is neither inherently good nor bad, but is conformant (or not) to a set of specific properties [4]. The SAAM method shows how to assess an architectural design along these lines [13]. Shaw et al. demonstrate that different software architecture models have different strengths and weaknesses and therefore lead to different architectural design solutions with significantly different software properties [22]. Therefore, software architecture models should be chosen in accordance with the system requirements and a software architecture model is suitable for a sub-set of properties. By “suitable” we mean that the model helps to either verify or assess a property. In this paper, we presented a set of properties that the Clover architectural metamodel verifies. In particular, the Clover metamodel accommodates style heterogeneity by allowing a pile of several layers with different Clover breakdowns, modifiability, reusability as well as extensibility.

Another avenue to validate an architectural model is to show that it corresponds to an implicit software practice: the model therefore makes explicit programmers savoir-faire. To do so, in [15], we studied the code of three groupware applications: a mediaspace developed in our team [6], a shared text editor NetEdit [25] and a collaborative ping-pong game [12]. By reverse engineering based on the code of the three applications, we showed that the modules of the existing code are organized according to our Clover metamodel.

Finally the generality of the metamodel (and in particular the variable number of layers and the possible partial Clover breakdown of a layer) is a required property for embracing the diversity and the novelty of the technical problems raised by groupware. Indeed the generality of an architectural model stems from its capacity to adapt to a variety of constraints according to a sound design rationale. So far the Clover metamodel has proven useful in triggering the right software design questions and in providing operational answers for several cases, both for forward design and engineering as well as for reverse design and engineering. On the one hand, the Clover metamodel guides the development of a future system such as CoVitesse; on the other hand it helps in understanding the organization of existing code. We do feel however that Clover metamodel patterns and heuristic rules need to be devised to respond to recurrent problems in multi-user systems. Indeed the unavoidable generality of the metamodel makes its application difficult. As done for our PAC-Amodeus software architectural model [17], providing patterns and rules will serve as guidelines for applying the Clover architectural metamodel.

CONCLUSION

We have presented a new conceptual architectural metamodel, the Clover architectural metamodel. The Clover metamodel results from the combination of the layer approach of Dewan's generic architecture with the functional decomposition of the Clover design model. The Clover architectural model has the following properties:

- By applying a Clover functional breaking up of each layer of our metamodel, we make system design issues explicit within the software architecture. Indeed the software architecture modeling is a design activity at the turning point between two worlds: the system design field and the programming field. Because of its interlinking location, software architecture must take into account design issues and properties of the two worlds.

- The Clover metamodel results from a motivated combination of existing architectural models selected for the complementarity of their properties. In particular our metamodel inherits the generic property from the layer approach of Dewan's generic architecture. The metamodel also refines each layer and event according to the three

Clover facets. This Clover breakdown of the layer increases the modularity of the code.

One architectural model derived from the Clover metamodel has been applied to the software design of the CoVitesse system. This derived model (Clover model), that has been presented in the paper and illustrated using CoVitesse, refines the Functional Core in terms of replicated and shared components. Moreover the Functional Core components are partitioned into three Clover sub-components dedicated to production, communication and coordination, the three sub-components being encapsulated by a wrapper.

Future developments consist of completing a Clover platform for developing groupware. The platform illustrates our Clover architectural metamodel by providing software components dedicated to the three Clover facets. The platform implements reusable encapsulated mechanisms that facilitate the implementation of any groupware. The CoVitesse system has both implemented using our platform. We are currently testing the robustness of the platform.

ACKNOWLEDGMENTS

This work has been partly supported by the French Ministers of Research and Industry Contract SIRII. Special thanks to G. Serghiou for help with style and English Grammar.

REFERENCES

1. Anderson, G., Graham, T.C.N., Wright, T., Dragonfly: Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems, in *Proceedings of ICSE'00*, 2000, p. 252-261, ACM Press.
2. Baecker, R.M., Nastos, D., Posner, L.R., Mawlby, M.K., The user-centered iterative design of collaborative writing, in *Proceeding of the Workshop on Real Time Group Drawing and Writing Tools, CSCW'92*, 1992.
3. Bass, L., and al., A Metamodel for the Runtime Architecture of an Interactive System, The UIMS Tool Developers Workshop, in *SIGCHI Bulletin*, vol. 24(1), 1992, p. 32-37, ACM Press.
4. Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, 1998, 452 pages, Addison Wesley.
5. Calvary, G., Coutaz, J., Nigay, L., From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW, in *Proceedings of CHI'97*, 1997, p. 242-249, ACM Press.
6. Coutaz, J., Bérard, F., Carraux, E., Astier, W., CoMedi: Using Computer Vision to Support Awareness and Privacy in Mediaspaces, in *Proceedings of CHI'99,* 1999, p. 13-14, ACM Press.
7. CoVitesse system, http://iihm.imag.fr/demos/CoVitesse/
8. Dewan, P., Architectures for Collaborative Applications, in Beaudouin-Lafon (eds.), *Computer Supported Cooperative Work*, 1999, p. 169-194, John Wiley & Sons Ltd.
9. Ellis, C., Wainer, J., A Conceptual Model of Groupware, in *Proceeding of CSCW'94*, 1994, p. 79-88, ACM Press.
10. Green, M., Report on Dialogue Specification Tools, in *User Interface Management Systems, Eurographics Seminars*, 1985, p. 9-20, Springer Verlag.
11. Graham, T.C.N., Urnes, T., Nejabi, R., Efficient distributed implementation of semi-replicated synchronous groupware, in *Proceedings of UIST'96*, 1996, p. 1-10, ACM Press.
12. java, http://www.javasoft.com/
13. Kazman, R., Bass, L., Abowd, G., Webb, M., SAAM: A Method for Analyzing the Properties of Software Architectures, in *Proceedings of ICSE'94*, 1994, p. 81-90, ACM Press.
14. Laurillau, Y., Synchronous Collaborative Navigation on the WWW, in *Extended Abtracts of CHI'99*, 1999, p. 308-309, ACM Press.
15. Laurillau, Y., Conception et réalisation logicielles pour les collecticiels centrées sur l'activité de groupe : le modèle et la plate-forme Clover, *Ph.D. dissertation*, University of Grenoble, France, 2002, 264 pages.
16. Li, D., Muntz, R., COCA: Collaborative objects coordination architecture, in *Proceedings of CSCW'98*, 1998, p. 179-188, ACM Press.
17. Nigay, L., Coutaz, J., Software architecture modelling: Bridging Two Worlds using Ergonomics and Software Properties, *Formal Methods in Human-Computer Interaction*, 1997, p. 49-73, Springer Verlag.
18. Nigay, L., Coutaz, J., A Generic Platform for Addressing the Multimodal Challenge, in *Proceedings of CHI'95*, 1995, p. 98-105, ACM Press.
19. Nigay, L., Vernier, F., Design Method of Interaction Techniques for Large Information Spaces, in *Proceedings of AVI'98*, 1998, p. 37-46, ACM Press.
20. Patterson, J.F., A taxonomy of Architectures for Synchronous Groupware Applications, in *Workshop on Software Architectures for Cooperative Systems of CSCW'94*, 1994, p. 79-88, ACM Press.
21. Roseman, M., Greenberg, S., Building Real-Time Groupware with GroupKit, A Groupware Toolkit, in *ACM ToCHI*, vol. 3(1), 1996, p. 66-106, ACM Press.
22. Shaw, M., Garlan, D., *Software Architecture: Perspectives on an Emerging Discipline*, 1996, 242 pages, Prentice Hall.
23. Salber, D., De l'interaction homme-machine individuelle aux systèmes multi-utilisateurs. *Ph.D. dissertation*, University of Grenoble, France, 1995, 305 pages.
24. Whittaker, S., Nardi, B., Bradner, E., Interaction and Outeracion Instant Messaging In Action, in *Proceedings of CSCW'00*, 2000, p. 79-88, ACM Press.
25. Zafer, A., NetEdit: a Collaborative Editor, *Master of Science*, University de Virginia, USA, 2001, 82 pages.