The AMODEUS Project
ESPRIT Basic ResearchAction 7040

Conceptual Software Architecture Models for Interactive Systems

Coutaz, J., Nigay, L. & Salber, D.
LGI-IMAG, Grenoble

20th March 1993

Amodeus Project Document: SystemModelling/WP11

Conceptual Software Architecture Models for Interactive Systems

Coutaz, J., Nigay, L. & Salber, D.
LGI-IMAG, Grenoble

Abstract

This document discusses architecture modelling for the software aspects of interactive systems. The literature shows a wide variety of such models revealing distinct goals, applicability, or scientific beliefs. In order to facilitate the presentation, a simple taxonomic framework is introduced based on the nature and the granularity of the entities that models use as well as on the extent to which models are able to support a particular set of requirements. The dimensions of the MSM framework are used as a set of such criteria to evaluate and compare two categories of architectural models: the conceptual multi-agent models such as GIO, PAC and MVC, and the implementational models such as Seeheim, Arch and PAC-Amodeus.

Keywords: GIO, PAC, MVC, Seeheim, Arch, PAC-Amodeus, MSM framework, multi-agent models, interactor, software architecture modelling, interactive systems.

Table of Contents

1. Introduction

This document discusses architecture modelling for the software aspects of interactive systems. The literature shows a wide variety of such models revealing distinct goals, usages, or scientific beliefs. In order to organize our current knowledge and to clarify our understanding about architectural issues, we propose a dimension space that may be useful for characterizing architectural models of interactive systems. This minimal 4-D space is presented in Section 2.

Most of architectural models are driven by the Graphical User Interface (GUI) paradigm. In parallel with the development of the GUI technology, significant progress has been made in natural language processing, computer vision and gesture analysis. Systems integrating these techniques as multiple modalities open a complete new world of experience [Krueger 90]. But as pointed out in [Blattner et al. 90], differences of opinion still exist as to the meaning of the term "multimodal". Working paper "Amodeus 7040, SystemModelling/WP4" defines the MSM framework for characterizing interactive systems including multimodal user interfaces. This design space, which adopts a system perspective, identifies issues that should be covered by software architecture studies. Section 3 summarizes the main points of the MSM framework.

Architectural models for interactive systems all rely on similar foundations. These fundamental principles are presented in Section 4. Sections 5 and 6 are dedicated to the description of significant architectural models in the field. Section 5 is concerned with the conceptual approaches to architectural issues whereas Section 6 presents models motivated by implementation issues. Whether they be concept-driven or implementation-driven, these models cover the organization of interactive systems considered as a whole.

The "holistic" models should be opposed to the specialized architectural models that focus on particular types of modalities and techniques such as computer graphics, computer vision and natural language processing. In a forthcoming Amodeus working paper, we will discuss the peculiarities and commonalities of the special purpose architectural models and show how they fit into the holistic frameworks.

2. A dimension Space for Architectural Models

An architectural model identifies entities and describes how these units relate to each other. The nature of the entities depends on the goal or the perspective adopted by the designers of the model. In particular, a software architecture model may be conceptual or aimed at implementation:

- within a conceptual model, an entity denotes a concept, for example the notion of dialogue control or that of an interactor [Duke et al. 92]. The model then shows how the concepts relate to each other in the design space;

- within an implementation model, an entity corresponds to a software component, for example the dialogue controller. The model then specifies the relationships between the components. Such relationships are driven by practical software engineering

considerations such as communication protocols and correspondence with reusable code such as libraries (e.g., Xlib and Motif).

Orthogonal to the perspective adopted, is the granularity of the entities: "How big is a chunk?" For example, what does the notion of dialogue control cover? What is an interactor? Can these notions be refined in terms of more elementary concepts?

Another interesting dimension is how well a model supports "quality criteria". Quality criteria are useful notions for evaluating a particular model against specific requirements. They can be subdivided into a number of ways to account for distinct classes of requirements. One category includes software engineering factors such as reusability and maintainability; another class would cover the notion of usability as described in [Abowd et al. 92] ; yet another one would characterize links with a theory as the theory of interactors investigated within Amodeus [Duke et al. 93]; finally, the dimensions of the MSM framework could also serve as a set of criteria for evaluating architectural models [Coutaz et al. 93a, 93b].

Whether it be conceptual or implementational, whether it be fine grained or not, whatever the quality criteria it is able to support, a model:

- may either serve as a guide during the software design process of an interactive system, or

- be used a posteriori as a reference to evaluate a particular software design or to reason about a particular solution within the design space provided by the model.

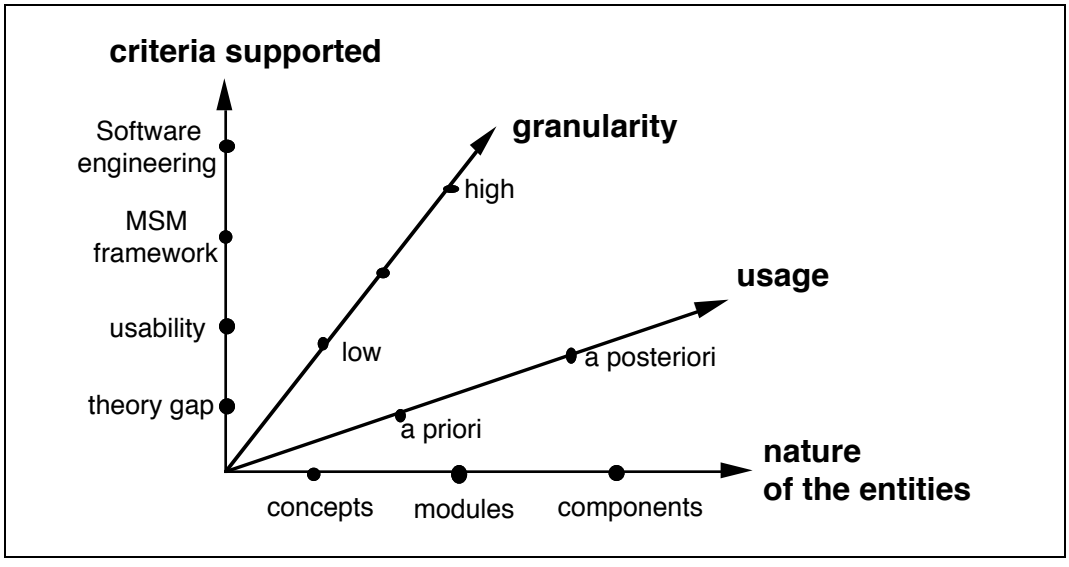

Figure 1: A minimal dimension space for characterizing software architecture models.

Figure 1 summarizes our minimal dimension space for characterizing software architecture models for interactive systems. It includes, the following four dimensions:

- the nature of the entities involved: do the entities represent concepts, software components, levels of abstractions, a combination of these, etc.?

- the granularity of the entities: what is the level of refinement supported by the model?

- the usage of the model: can it be used as a driving guide during the design process of an architectural solution, or should it be used as a reference to assess the soundness of a particular architectural design?

- the quality criteria: how good is the architectural model to support quality requirements? As discussed above, quality requirements illustrate multiple perspectives from software engineering to HCI factors, and theoretical issues. Following on the MSM design framework developed within Amodeus2, we will evaluate architectural models against the MSM dimension.

3. Brief Overview of the MSM framework

The MSM framework is a dimension space that should help reasoning about current and future Multi-Sensori-Motor systems (MSM). As shown in Figure 2, this problem space is comprised of 6 dimensions. The first two dimensions deal with the notion of communication channel: the number and direction of the channels that a particular MSM system supports. A communication channel covers a set of sensory (or effector) means through which particular types of information can be received (or transmitted) and processed.

The other four dimensions of the MSM framework are used to characterize the built-in cognitive capabilities of the system: the interpretation and the rendering functions. The interpretation function covers the sequence of transformations applied to inputs received through sensors. In the other direction, internal information (e.g., the system state) is transformed by the rendering function to make it perceivable to the user via effectors. The interpretation and the rendering functions are both characterized by four intertwined ingredients: level of abstraction, context, fusion/fission, and parallelism.

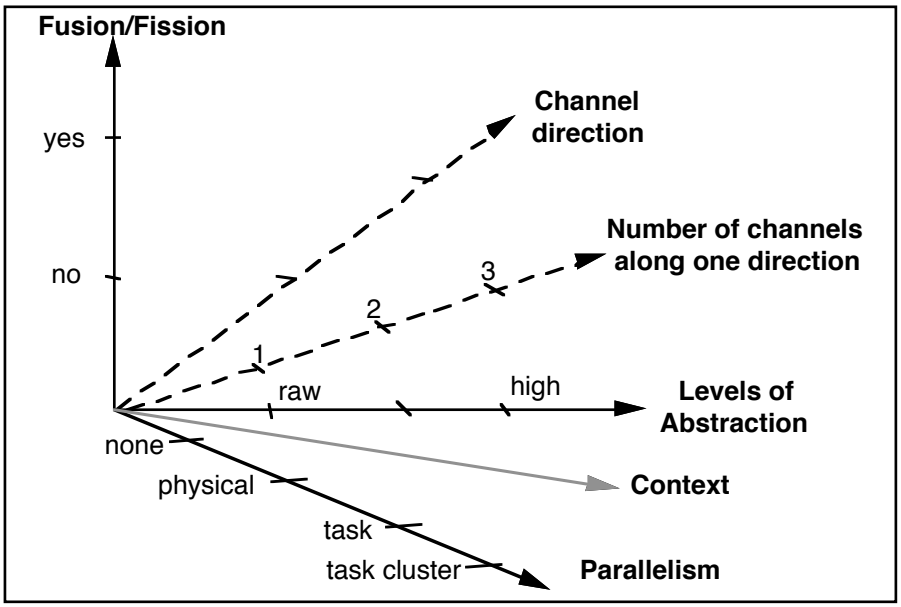

Figure 2: The MSM Framework: a 6-D space to characterize multi-sensory-motor systems.

The level of abstraction expresses the degree of transformation that the interpretation and rendering functions perform on information. It also covers the variety of representations that the system supports, ranging from raw data to symbolic forms. For a given digital input channel, the interpretation function can be characterized by its power of "abstracting" raw data into higher representational expressions. The rendering function is characterized by the level of abstraction it starts from to produce perceivable raw information through output digital channels (concretization phenomenon).

The capacity of a system to abstract or to concretize along a channel may vary dynamically with respect to "contextual variables". Contextual variables are like cognitive filters. They form a set of internal state parameters used by the representational processes to control the interpretation/rendering function. Context covers the notion of mode as defined by Thimbleby [Thimbleby 90].

Fusion refers to the combination of several chunks of information to form new chunks. Fission refers to the decomposition phenomenon. Fusion and fission are part of the abstracting and concretization phenomena.

Parallelism at the interface may appear at multiple grains: at the physical level, at the elementary task level (i.e., the command level), and at the task level. For input, parallelism at the physical level allows the system to probe the user's physical actions through multiple sensors simultaneously. For output, the system is able to express a state change through multiple effectors simultaneously. Parallelism at the elementary task level allows the user to express multiple commands simultaneously or in an interleaved way. A task is a cluster of tasks that structures the interaction space. Parallelism at the task level is, in general, supported through interleaving between different clusters or via true parallelism between commands that belong to distinct clusters.

Having defined a minimal dimension space for reasoning about software architecture models, we are now able to present a number of holistic architecture models and analyze them along this framework including the MSM framework.

4. The Foundations of Holistic Models

All of the holistic architectural models have adopted the distinction of concerns between the entities that model the task domain and those involved in the perceivable portion of the system.

From a conceptual linguistic perspective, this distinction corresponds to the notions of semantics and pragmatics on one hand, and the syntactic, lexical and articulatory issues on the other hand. From a software engineering perspective, the distinction between task domain and interpretation/rendering opens the way to code re-usability and code maintainability. In particular, the entities that depend on the task domain define a software component, the functional core, that can be reused with different interpretation and rendering functions. Similarly, interpretation/rendering entities define a software component, the user interface, that can be maintained without modifying the functional core. The identification of these two components has helped in the emergence of the UIMS technology.

Figure 3 shows a software interpretation of the conceptual consensus. It is characterized by a coarse granularity of the components, and the model can be used during the design phase as well as for evaluation. With regard to software quality criteria, it clearly stresses maintainability and iterative design. As far as the MSM criteria are concerned, the foundation model:

- implicitly supports the notion of communication channels within the user interface,
- explicitly identifies two levels of abstraction: the high level exemplified by the functional core and the low level within the user interface,
- says nothing about context, parallelism and fusion/fission phenomena.

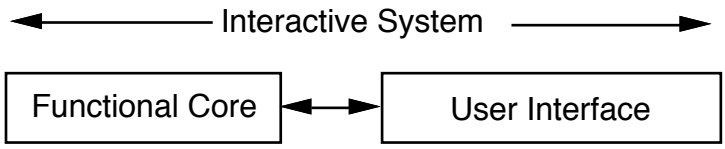

Figure 3 : The foundation model suited to software architecture modelling.

In summary, the conceptual framework which introduces two notions (task domain and user interface), has opened the way to a number of interpretations. One of them, adopted by the software engineering community, is a 2-component model that supports the software engineering principles of maintainability and reusability. Although the foundation model is characterized by a coarse granularity, the concepts it introduces have paved the way 1) to conceptual architectural models, such as MVC [Krasner et al. 88], PAC [Coutaz 87], GIO [Faconti 93, Paterno 93] and the Lisbon model [Duce et al. 91], and 2) to implementational models such as Seeheim [Pfaff 85], Arch [Arch 92], and PAC-Amodeus [Nigay et al. 91b].

4. Conceptual Architectural Models: MVC, PAC, GIO, Lisbon

MVC, PAC, GIO, and the Lisbon model introduce two additional features to the notions of task-domain and user interface: the concept of concurrency and that of level of abstraction. These notions are conveyed by a set of cooperating agents, interactors, and objects.

In this section we clarify the relations between the concepts of agent, interactor, and object. Then we show how agents support the separation of concerns with the notion of perspective. We close the section with the evaluation of MVC, PAC, GIO and Lisbon with regard to the MSM framework.

4.1. Agents, Interactors, Objects

An agent is a complete information processing system: it includes event receivers and event transmitters, a memory to maintain a state, and a processor which cyclically processes input events, updates its state, and may produce events or change its interest in input event classes. Thus an agent is a unit of competence which operates in parallel and in a coordination with other agents. The user is such an agent. A formal definition of the notion of agent can be found in [Abowd 91].

Our view of the concept of agent is one perspective of the more general definition used in distributed Artificial Intelligence (A.I.). In A.I., agents may be cognitive or reactive depending mainly on their reasoning and knowledge representation capabilities [Demazeau 91]. A cognitive agent is enriched with inference and decision making mechanisms to satisfy goals. At the opposite, a reactive agent has a limited computational capacity to process stimuli. It has no goal per se but a competence coded (or specified) explicitly by the human designer. In the following discussion, we will not make the distinction between cognitive and reactive agents although current models and formalisms developed for the software design and verification of user interfaces consider reactive agents only.

Interactors are the agents of an interactive system that communicate with the user directly. They provide the user with a perceptual representation of their internal state [Duke et al. 92, Duke et al. 93] as well as a means to modify this state. Interactors are also coined as "interaction objects".

An object is a generic term that covers the notion of computing entity with a local state. It can either be viewed as a concept or as the technical structure that underpins the object-oriented programming paradigm. In the following discussion, we will consider an object as a generic concept. An agent is a kind of object. Unlike some objects that may be passive (i.e., manipulated only), an agent is an active object. The Arch and PAC-Amodeus models presented in Section 5 illustrate the combined presence of passive and active objects.

In summary, an interactor is an agent that interacts with the user directly. An agent is an active object, and an object is a state-based process. The following expression clarifies the relationships between the sets of objects, agents and interactors:

an interactor *is-an* agent *is-an* object[1]

The conceptual models described in this paper all view an interactive system as a set of cooperating agents. In addition, MVC, PAC and GIO[2] refine agents into a number of perspectives.

4.2. Agents, Perspectives and Separation of Concerns

Basically, perspectives within an agent correspond to the notions introduced by the foundation model: task domain and user interface.

In MVC, an agent is modelled along three perspectives: the Model, the View, and the Controller. A Model defines the functional competence of the agent (i.e., its task-domain). The View corresponds to the rendering function. It defines the perceivable behavior of the

---

[1] Another terminology could be adopted: an agent would cover the A.I. definition and an interactor would be the reactive AI agent. We need a name to denote interactors that communicate with the user directly. (I/O object and logical device are not satisfactory: an object may be passive and, the definition of logical device covers the motivations behind the notion of interactor.)

[2] "GIO" stands for Graphical Interaction Object as defined by Paterno in [Paterno 93]. We have introduced the term "GIO model" to denote the approach described by Faconti and Paterno. We hope that we do not make injustice to their work by doing so!

agent. The Controller denotes the interpretation function of inputs. The View and the Controller define the user interface of the agent, i.e., its behaviour with regard to the user.

PAC conveys similar ideas: a PAC agent has a Presentation perspective (its rendering and interpretation functions), an Abstraction (its task domain competence), and a Control. The Control is in charge of communicating with other agents as well as bridging the gap between its abstract and user interface facets.

In GIO, an agent possesses four perspectives : the Collection and the Abstraction facets define the "functional core" aspect of the agent for input and output respectively; the Measure and the Presentation cover the user interface side of the agent for input and output respectively.

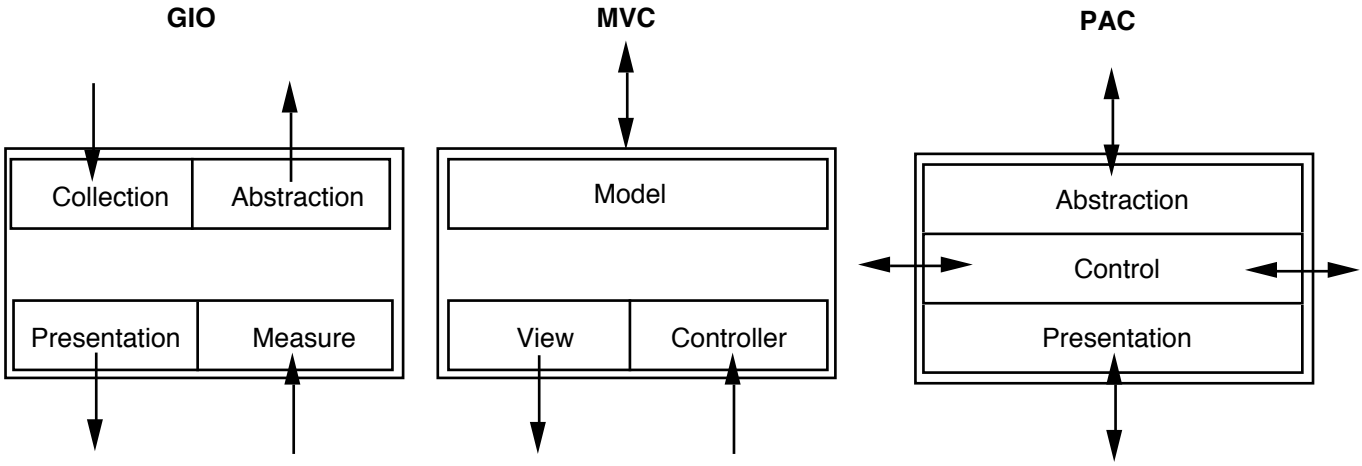

Figure 4: Equivalence between the perspectives of MVC, PAC and GIO agents. Arrows indicate the directions of inputs and outputs.

Figure 4 summarizes the commonalities and the differences between MVC, PAC and GIO:

- an MVC "View" is equivalent to a GIO "Presentation".
- an MVC "Controller" is equivalent to a GIO "Measure".
- an MVC "Model" is equivalent to a PAC "Abstraction" is equivalent to a GIO "Abstraction, Collection" couple.
- an MVC "View, Controller" couple is equivalent to a GIO "Presentation, Measure" couple is equivalent to a PAC "Presentation".
- a PAC "Control" has no explicit correspondence with the MVC facets. With regard to GIO, it seems to cover the internal coordination between the components of an agent as well as the notion of "Controller" introduced in [Faconti 93] to control the behavior of an agent.

In the Lisbon model, agents do not have perspectives but are organized as specialized categories to model the rendering and interpretation functions.

4.3. Conceptual multi-agent models and the MSM Framework

In this paragraph, the discussion is organized along the dimensions of the MSM framework: communication channels, levels of abstraction, context, fusion and fission, and parallelism.

4.3.1. Communication channels

All of the models considered in our discussion stress the fact that an agent may communicate with multiple agents. Agents are multichannel capabilities. However, not all of the multi-agent models make explicit the communication channels their agents support. For example, MVC does not say how communication occurs between the agents of an interactive system. The question is left opened until implementation.

In PAC, on the other hand, the Control facet of an agent supports communication in two ways: first, it serves as an explicit bridge between the two facets it serves (the Abstraction and the Presentation may use different formalisms as well as distinct time basis); second, it is used as the switchboard of the agent: it receives inputs and outputs from other agents. At the opposite of GIO, however, PAC does not make explicit how inputs and outputs are processed and dispatched within the agent.

In GIO, input and output channels are clearly expressed. For example, the Measure of an agent is modelled as the local process specialized in processing inputs from lower level agents. The Collection has a similar role for processing inputs from higher levels of abstraction. The Presentation and the Abstraction are the output channels for lower levels and higher levels of abstraction respectively.

4.3.2. Levels of Abstraction

Information acquired by interactors is transformed by a population of agents before reaching the functional core. This transformation process, called the interpretation function in the MSM framework, defines the capacity of abstracting of the user interface. In the other direction, the rendering function corresponds to the capacity of the user interface to concretize information from to functional core into the formalism acceptable for interactors. The successive steps of such transformations define levels of abstraction. In the conceptual architectural models considered in our discussion, abstracting and concretizing are performed by agents organized into levels of abstraction.

Within GIO, levels of abstraction are defined in two ways: within an agent and in the form of a composition mechanism.

- A GIO agent stresses a clear distinction between the rendering and the interpretation functions. These functions operate along a 2 step process. Each step defines a level of abstraction within the agent. For rendering, information from higher levels of abstraction is received by the Collection and delivered to lower levels by the Presentation. For interpreting, the Measure receives information from lower levels whereas the Abstraction communicates with higher levels.

- GIO agents can be composed into more powerful, abstract agents with composition operators such as the interleaving operator [Faconti 93]. By doing so, an agent is defined as a hierarchy of agents composed of a parent and a set of interleaved child agents.

PAC and MVC adopt a similar approach for modelling levels of abstraction. Like GIO, they do not make explicit the nature of these levels. The Lisbon model, on the other hand, introduces classes of agents, i.e. Conceptual objects (CO's), Interaction Objects (IO's), and Transformer Objects (TO's) as an indication of what these levels might be.

In the Lisbon model, CO's are "the user interface accessible representation of an object the functional core wishes to make visible" [Duce et al. 91]. In the user interface portion of an interactive system, CO's are the proxies of the task domain concepts manipulated by the functional core. IO's support the translation process between CO's inputs and outputs and low level input and output devices. As in GIO, "Composite IO's may be formed from single IO's to handle arbitrarily complex threads of dialogue" [Duce et al. 91]. TO's provide the basic mechanism for managing the relations between different objects of the user interface. Examples of such relations include constraint management to maintain consistency between IO's, context switching between dialogue threads, integrity mapping between CO's and IO's, etc.

4.3.3. Context

The MSM framework does not provide a sound definition of the notion of context. However, an agent, which is a state-based processing capability, illustrates one possible class of context.

4.3.4. Fusion and Fission

The composition mechanism within MVC, PAC, and GIO was primarily introduced for the expression of levels of abstraction. One side effect of this mechanism is that composition provides a sound foundation for fusion and fission. Fusion and fission are part of the internal processing capabilities of an agent.

4.3.5. Parallelism

As stated above, an agent is a processing unit that can operate in parallel with other agents. It results from multi-agent architectures that multiple interpretation and rendering activities can take place simultaneously.

4.4. Summary

In summary, if we refer to our simple classification scheme of Section 2, MVC, PAC, GIO and the Lisbon models are all driven by conceptual considerations. With regard to granularity, all of them manipulate the notion of agent although they refine an agent in different ways. As shown in Figure 4, PAC, MVC and GIO differ by the granularity of the description of an agent. GIO, which explicitly decomposes the abstract and the concrete sides of an agent into two facets appears as the more fine grained model. PAC, on the contrary is more macroscopic.

MVC, PAC, GIO and the Lisbon model, all pass the MSM criteria successfully. They all provide conceptual mechanisms for the MSM dimensions as well as for the foundation principle:

- distinction between concrete rendering from abstract functionality,
- support for parallel activities and multiple I/O channels,

- contextual transformation of information from low level devices to high level task-domain concepts and vice versa (abstraction and concretization).

Although they support similar mechanisms, multi-agent models differ on the focus and the means:

- The Lisbon model expresses the abstraction/concretization process by introducing specialized agents. In MVC, PAC and GIO, these transformations are performed by unmarked agents.

- Whereas PAC, MVC and GIO stress the distinction between rendering issues and conceptual functionalities at multiple levels of abstraction, these notions are blurred within the Lisbon model.

- Contrary to PAC and the Lisbon model, MVC has no explicit notion of arbitrator for expressing the relationships and the coordination between the agents.

Because they are conceptual, MVC, PAC, GIO, and the Lisbon model do not impose any implementation technique. However they can be easily expressed with object-based programming languages. For example, within the Smalltalk environment [Goldberg 84], an MVC agent class is implemented as three Smalltalk classes (one class per facet) that are connected through explicit message passing. Depending on the programming tool at hand, a PAC agent may be implemented either as one module, or a single object class, or even as in Smalltalk as a cluster of three classes. One can refer to [Paterno 93] for the implementation of GIO agents in terms of an object-oriented language.

The Seeheim, Arch, and PAC-Amodeus models presented in the next section are more concerned with implementation issues than the multi-agents models considered above.

5. Implementational Architectural Models: Seeheim, Arch, PAC-Amodeus

5.1. Seeheim

The term “Seeheim model” originated at a workshop in Seeheim, Germany [Pfaff 85]. As shown in Figure 5, this model refines the user interface portion of an interactive system into three components: the Application Interface[3], the Dialogue Control and the Presentation. The role of each component is roughly described as the semantic, syntactic and lexical functionalities of the user interface. The Application has the same definition as the Functional Core: it models the domain-specific concepts (i.e., the semantics), whereas the Dialogue Control and the Presentation deal with syntactic and lexical issues respectively. The Application Interface defines the view that Dialogue Control has about the Application. If we refer to the Lisbon model, CO objects may be gathered within that component.

[3] “Application Interface” and “Functional Core Interface” are equivalent expressions.

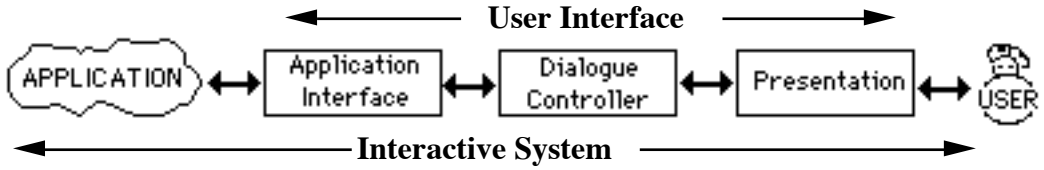

Figure 5: The Seeheim Model.

The Seeheim model is based on notions (semantics, syntax, and lexicon) which are well understood by computer scientists. Therefore, it is a useful framework for teaching how to devise the design process of an interactive system as well as how to organize its software architecture. Design methods such as CLG [Moran 81] and that of Mark Green [Green 85] and Foley and Van Dam [Foley 84], which advocate a top-down design approach (from the definition of the domain concepts to the identification of the interaction techniques used in the presentation) are consistent with the linguistic interpretation of the Seeheim model. With regard to software tools, the Seeheim model has opened the way to the transfer of knowledge from automatic compiler generation to automatic user interface generation. The result has been an emergence of a wide variety of rapid prototyping tools.

Unfortunately, experience has shown that user interface designs based on the linguistic approach presented major limitations. By nature, a language view puts the emphasis on the form, and is totally oblivious to the dynamics. This prejudice is acceptable for a compiler which processes fully specified static input expressions.

As expressed by the MSM framework, the form of an input expression may evolve in parallel with the production of an output expression; multiple input expressions may be specified simultaneously. Unlike a compiler which performs a well-defined sequence of processing, the user interface must, in general, support parallelism. In addition, in order to augment the quality of domain-specific feedback, it should allow for partial input expressions to be interleaved with the production of output expressions; it should support a flexible granularity for input and output expressions; it should support multithread dialogues in order to allow the user to simultaneously handle several threads of reasoning as well as to behave in an opportunistic manner [Hayes-Roth 79].

Thus, applying a centralized sequential linguistic view on top of the Seeheim model, makes impossible the satisfaction of some of the most fundamental user-centered requirements. In addition, the Seeheim model does not provide software designers with any help to perform the unavoidable and difficult engineering trade-offs. Such trade-offs affect the optimization of the development process as well as the quality of the end-product. The Arch model and its companion, the slinky metamodel, aim at filling the gap.

5.2. The Arch Model

The Arch model results from the work of the user interface tool developers group [Arch 92]. Its purpose is to provide developers with a framework for understanding engineering trade-offs. It is not intended to be prescriptive but rather usable as a reference for evaluating a particular candidate run-time architecture. The authors observe that “user interface developers sometimes find both the application domain functionality[4] and the User Interface

[4] “Application domain functionality”, “Application”, and “Functional Core” are equivalent expressions.

Toolkit(s) to be existing constraints upon the development of a user interface... For this reason, the domain software and the User Interface Toolkits form the two bases of the Arch model." [Arch 92, pp. 34]. As shown in Figure 6, the Arch model is a refinement of the Seeheim model where engineering reality has been injected. In addition, the Arch model makes explicit the nature of the information that cross the boundaries between the components.

5.2.1. The Components of the Arch Model

The Domain-Specific Component and the Domain Adaptor Component are different terms for denoting the notions of Application and Application Interface introduced by Seeheim. Architecture modelers however, have a better understanding of the role of the Domain Adaptor[5] as well as on the nature of the data exchanged between the Domain-Specific Component and the Domain Adaptor. These issues will be further discussed in a later paragraph about Adaptor Components.

The Dialog Component corresponds to the Seeheim Dialogue Controller although in Seeheim, the role of this component was limited to some obscure syntactic sequential processing such as the combination of lexical inputs into command level abstractions. In the Arch model, the Dialogue Component "has responsibility for task-level sequencing, ..., for providing multiple view consistency and for mapping back and forth between domain-specific formalisms and user-interface-specific formalisms" [Arch 92, p. 34].

The domain-specific formalism describes the entities that are exchanged with the Domain Adaptor whereas the user-interface-specific formalism expresses semantically identical information driven by presentation considerations. For example, the notion of temperature in the Domain Adaptor would be represented as a real number whereas, at the presentation level, it would be represented as the mercury height of a thermometer. The first formalism, i.e., the real number, is suggested by computational purpose whereas the second formalism, a mercury height, depends on the formalism provided by the Presentation component (e.g., graphics, voice).

The Arch model segments the Seeheim Presentation into two levels of abstraction: The Presentation Component and the Interaction Toolkit Component. The Interaction Toolkit Component implements the physical interaction with the end-user. The Presentation Component "provides a set of toolkit-independent objects for use by the Dialogue Component ..." [Arch 92, p. 34]. A toolkit-independent object is called a "presentation object" whereas "interaction objects" refer to interaction techniques supplied by interaction toolkits. For example, a "selector" presentation object can be implemented in the toolkit using either a menu or radio buttons interaction objects. Therefore, the Presentation Component acts as the second software adaptor of the Arch model.

[5]The "Domain Adaptor" is also called the "Functional Core Adaptor" by Nigay and Coutaz (Nigay & Coutaz, 1991b).

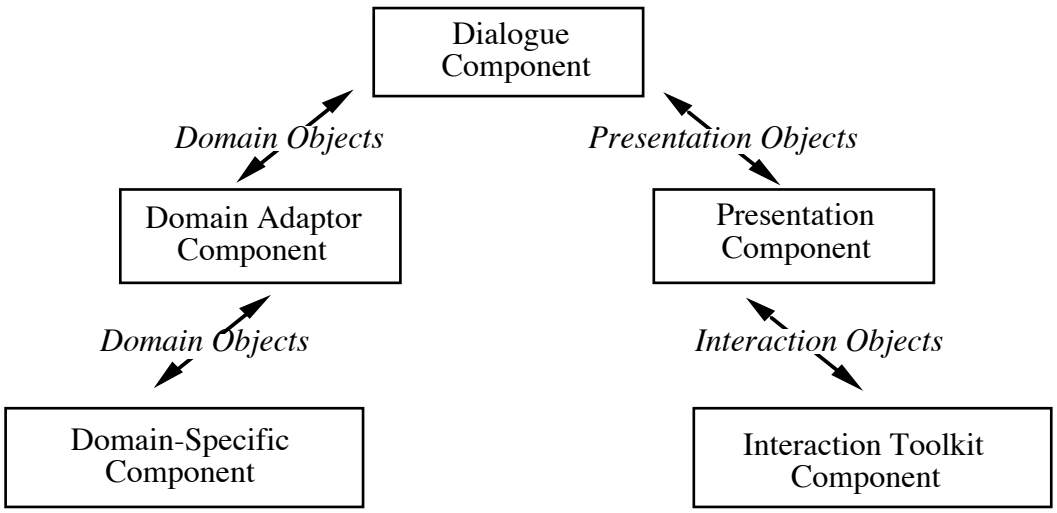

Figure 6: The components of the Arch model and their interfaces. [Arch 92]

5.2.2. The Adaptors of the Arch Model

Adaptor components, which define abstract interface machines, are useful concepts to improve code reusability, portability, and modifiability. This subsection discusses two instances of such adaptors in user interface software engineering: the Domain Adaptor and the Interaction Toolkit Adaptor, which insulate the keystone component (i.e., the Dialogue Component) from modifications in its mandatory neighbors: the Functional Core and the Interaction Toolkit.

The Domain Adaptor Component. The Domain Adaptor Component serves as a mediator between the Dialogue Component and the domain-specific concepts implemented in the Functional Core [Nigay 91b]. It is designed to absorb the effects of change in its direct neighbors. As any boundary, it implements a protocol. A protocol is characterized by temporal strategies and by the nature of data exchanged [Coutaz 91].

A temporal strategy defines the coordinating rules for transfering information between two communicating entities such as the Functional Core and the Dialogue Component. The coordination may be fully synchronous or fully asynchronous or may alternate between the two techniques. Synchronous coordination implies that the sender waits for the receiver before its own processing can resume. In the context considered here, synchronous coordination models the mutual control of the Functional Core and the Dialogue Component: either one has the initiative, but the initiator is directly controlled by its partner. Asynchronous coordination allows communicating entities to exchange information without waiting for each other. With such a communication scheme, the Functional Core and the Dialogue Component are two equal partners sharing a common enterprise: that of accomplishing a task with the user.

When considering interaction styles, synchronous coordination results in single threads of dialogue or, at best, in interleaved threads. Asynchronous coordination supports multiple concurrent threads of dialogue: the user may issue multiple commands simultaneously (using for example, a combination of voice and hands) while the Functional Core may have its own processing going on.

Exchange of data between the Functional Core and the user interface is performed through the Domain Adaptor in terms of domain objects[6]. A domain object is an entity that the designer of the Functional Core wishes to make perceivable to, and manipulable by the user. Ideally, it is supposed to match the user's mental representation of a particular domain concept. It may be the case however that the Functional Core, driven by software or hardware considerations, implements a domain concept in a way that is not adequate for the user.

Semantic enhancement [Bass et al. 91] may be performed in the Domain Adpator by defining domain objects that reorganize the information modeled by the Functional Core. Reorganizing may take the form of aggregating data structures of the Functional Core into a single domain object or, conversely, segmenting a concept into multiple domain objects. It may also take the form of an extension by adding attributes and operators, which can then be exploited by the other components of the user interface. In Serpent [Bass et al. 91], the Domain Adaptor is implemented as a data base of passive objects. The data base itself is an abstract machine whose operator allow the Functional Core and the Dialogue Component to collect, create, or destroy entities, but these entities, which model the state of the data base, are passive objects.

The Presentation Component. The Arch Presentation Component acts as a mediator between the Dialogue Component and the interaction toolkit. As shown in Figure 7, Coutaz proposes to refine the Presentation Component into two layers of abstraction: the Extension Layer and the Interaction Toolkit Adaptor [Coutaz 91]. The Interaction Toolkit Adaptor defines a virtual toolkit used for the expression of presentation objects. This expression is then mapped into the formalism of the actual interaction toolkit used for a particular implementation. Switching to a different toolkit requires rewriting the mapping rules, but the expression of the presentation objects remains unchanged.

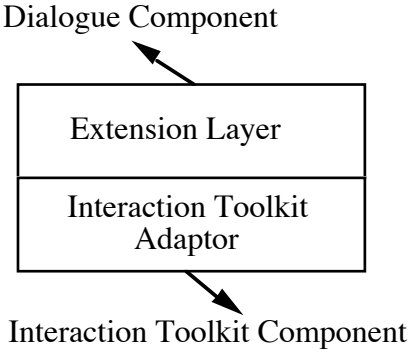

Figure 7: Refinement of the Presentation Component. (Coutaz, 1991)

Interaction objects are generally constructed from entities made available in interaction toolkits. In general, interaction toolkits such as X Intrinsics [OSF 89], provide an abstraction mechanism for defining new interaction objects. However, it is not always possible to build new interaction objects from the predefined building blocks of the toolkit. For example, in an earlier version of X intrinsics, widgets (i.e., interaction objects) would occupy rectangular areas only. Under such conditions, the notion of a wall in a floor plan drawing editor could

[6]The term "object" should be considered here in the general sense as an entity. It does not suppose any particular implementation technique such as a class, a function, or a shared data structure but covers all of them.

not be implemented as a diagonal line widget. Instead, a presentation object "wall" would have to be defined as a new presentation object outside the toolkit.

The wall example shows that the Presentation Component should, conceptually, be structured into two layers. Specific-interaction objects, which can be built from the building blocks of the toolkit, should belong to the toolkit. Those which cannot be built with the toolkit should be part of the Extension Layer. The Interaction Toolkit Adaptor, such as XVT [Valdez 89], defines the boundary between these two layers.

Sometimes, the location for implementing a presentation object is not straighforward. For example, the thermometer used to present the notion of temperature could be implemented either at the toolkit layer or as part of the extension layer. If located in the toolkit, then the thermometer becomes a general purpose interaction object and thus, should be implemented according to the programming rules imposed by the toolkit to guarantee reusibility. If located in the extension layer, the private status of the presentation object relaxes the reusibility constraints of the underlying platform.

The thermometer example is merely a simple illustration of a more general problem: that of identifying the appropriate location for software functionalities. The Slinky metamodel is an attempt to provide an answer to this difficulty.

5.3. The slinky metamodel

Experience shows that no single architectural model can satisfy all of the software design factors and criteria such as those reported by McCall [McCall 77]. Factors and criteria may be conflicting. For example, portability and modifiability may impede efficiency. In the case of the Arch model, two adaptor components have been introduced to minimize dependence on Functional Core modifications as well as on the effective user interface toolkit. Conversely, these abstract machines may have an adverse effect on the speed of the run-time end product.

Another example of conflict occurs between the modular distribution of functionalities and response time. The software foundations illustrated in Figure 3 stress that domain-specific objects should be confined in the Functional Core and the Functional Core Adaptor. It results from this principle that the semantic quality of feedback for a single user-system transaction may require many round trips between the user interface portion of the system and the Functional Core. This long chain of data transfer between the components of the user interface to reach the Functional Core and vice versa may be costly with respect to the system response time. Therefore, it may be inconsistent with the expectation of the user.

Domain-knowledge delegation[7], which consists of down-loading functional core knowledge into the user interface is a way to reduce transmission load at critical points [Coutaz 91] and thus to improve response time when this criteria has been identified as an important requirement. For example, rubber-banding in direct manipulation interfaces, requires high performance at the user interface. In particular, if the Functional Core is implemented as a distinct process running on a distinct processor, it may be judicious to delegate domain

[7]The term “semantic delegation” was previously used by Coutaz to denote “domain-specific delegation”.

knowledge into the user interface portion of the interactive system. By doing so, semantic knowledge is readily available in the user interface and can be rendered to the user within the response time constraint. Communication with the Functional Core can be postponed when response time is not critical.

The Slinky metamodel acknowledges the fact that software architectures for interactive systems must be tailored to the requirements and criteria selected for the particular case at hand. “The term “Slinky” was selected to emphasize that functionalities can shift from component to component in an architecture depending on: - the goals of the developers, - the weighting of development criteria, -the type of system to be implemented. This concept is loosely represented by the flexible Slinky™ toy.” [Arch 92, pp. 35].

Thus, the Slinky metamodel is a generic framework from which particular instances of Arch models can be derived. For example, if efficiency prevails against toolkit portability, then the Interaction Toolkit Adaptor may not be needed. If high quality semantic feedback and efficiency are important requirements as in semantically rich rubber-banding tasks, then domain-knowlege delegation may be performed. Such decision may result in reducing the relative importance, thus the code size, of the Functional Core. The PAC-Amodeus model presented in next Section provide a way to perform domain-knowledge delegation without jeopardizing the basic “separation of concerns” principle illustrated in Figure 3.

5.4. PAC-Amodeus

PAC-Amodeus [Nigay 91b], based on early experience with PAC, is a blend of the components advocated by Arch and the refining process in terms of agents advocated by PAC.

PAC-Amodeus adopts the same components as Arch and assigns the same roles as Arch to these components: the Functional Core corresponds to the Domain-Specific Component, the Functional Core Adaptor is the Arch Domain Adaptor Component, the Dialogue Controller denotes the Dialogue Component, and the Presentation Component embeds the two levels of abstraction shown in Figure 7. PAC-Amodeus goes one step further than Arch by decomposing the Dialogue Component into a set of cooperative PAC agents.

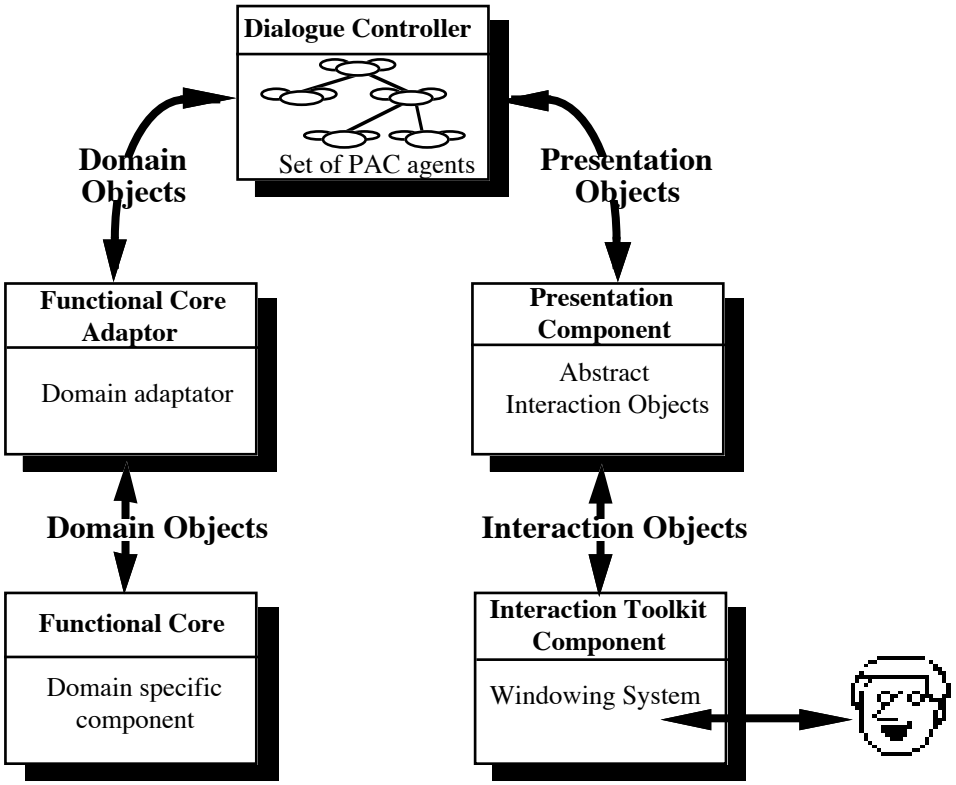

Figure 8: The PAC-Amodeus model. [Nigay 91b]

As presented above, the Dialogue Controller has the responsibility for task-level sequencing. Each task or goal of the user corresponds to a thread of dialogue. This observation suggests the choice of a multi-agent model. Indeed, a multi-agent model distributes the state of the interaction among a collection of cooperating units. Modularity, parallelism, and distribution are convenient mechanisms for supporting multi-thread dialogues. One agent or a collection of cooperating agents can be associated to each thread of the user's activity. Since each agent is able to maintain its own state, it is possible for the user (or the functional core) to suspend and resume any thread at will.

The Dialogue Controller receives events both from the Functional Core, via the Functional Core Adaptor, and from the user via the Presentation Component. Bridging the gap between a Functional Core Adaptor and Presentation Component has some consequences: In addition to task sequencing, the Dialogue Controller must perform data transformation and data mapping:

1) A Functional Core Adaptor and a Presentation Component use different formalims. One is driven by the computational considerations of the Functional Core, the other is toolkit dependent. In order to match the two formalisms, data must be transformed inside the Dialogue Controller.

2) State changes in the Functional Core Adaptor must be reflected in the Presentation Component (and vice versa). Therefore, links must be maintained between domain objects of the Functional Core Adaptor and presentation objects in the Presentation Component. As discussed in [Coutaz 91], a domain object may be rendered with multiple presentation techniques. Therefore, consistency must be maintained between the multiple views of the conceptual object. Such mapping is yet another task of the Dialogue Controller.

Thus, bridging the gap between the Functional Core Adaptor and the Presentation Component covers task sequencing, formalism translation, and data mapping. Experience shows that these operations must be performed at multiple levels of abstraction and distributed among multiple agents. Levels of abstraction reflect the successive operations of abstracting and concretizing. Abstracting combines and transforms events coming from the presentation techniques into higher level events for higher abstractions. Conversely, concretizing decomposes and transforms high level information into low level information. The lowest level of the Dialogue Controller is in contact with the presentation objects.

As discussed above, the multi-agent approach is a promising way to support parallelism, distribution, multithread dialogues, and iterative design. Since agents should carry task sequencing, formalism transformation, and data mapping at multiple levels of abstraction, it is tempting to describe the Dialogue Controller at multiple grains of resolution combined with multiple facets. At one level of resolution, the Dialogue Controller appears as a "fuzzy potato". At the next level of description, the main agents of the interaction can be identified. In turn, these agents are recursively refined into simpler agents. This description is nothing more that the usual abstraction/refinement paradigm applied in software engineering.

Orthogonal to the refinement/abstraction axis, we introduce the "facet" axis. An agent is also described along three facets: Presentation, Abstraction, Control. These facets are used to express different but complementary and strongly coupled computational perspectives.

- The Presentation facet of an agent implements the perceivable behavior of the agent. As shown in Figure 9, it is related to some presentation object of the Presentation Component.

- The Abstraction implements the competence of the agent (i.e., its expertise) in an essentially media-independent way. It is the Functional Core of the agent. It maintains the abstract state of the agent. It may be related to some domain object(s) of the Functional Core Adaptor. The abstraction facet of an agent provides a good mechanism for performing domain-knowledge delegation.

- The Control part of an agent is in charge of two functions: linkage of the Abstraction part of the agent to its Presentation portion and maintenance of the relationships of the agent with other agents. The linkage serves two purposes: 1) formalism transformations between the Abstraction and the Presentation portions of the agent, and 2) data mapping between the abstract facet and the presentation facet. Relationships between agents may be static or dynamic. Dynamic relationships are required when agents are dynamically created/deleted. Relationship maintenance by the control part of an agent covers the communication and the synchronization mechanism between this agent and its cooperating partners.

In summary, an agent could be viewed as a mini-Arch. Figure 9 shows how a PAC agent relates to other PAC agents and to the surrounding world of the Dialogue Controller: domain objects in the Functional Core Adaptor and presentation objects in the Presentation Component. If we consider the Dialogue Controller as a whole:

- the set of Abstraction parts of the various agents defines the internal state of the interaction

- the set of Presentation parts defines the external state of the interaction
- the set of Control parts defines the mapping functions between the internal and the external state. Some properties of these functions are defined in Harrison and Thimbleby [Harrison et al. 90].

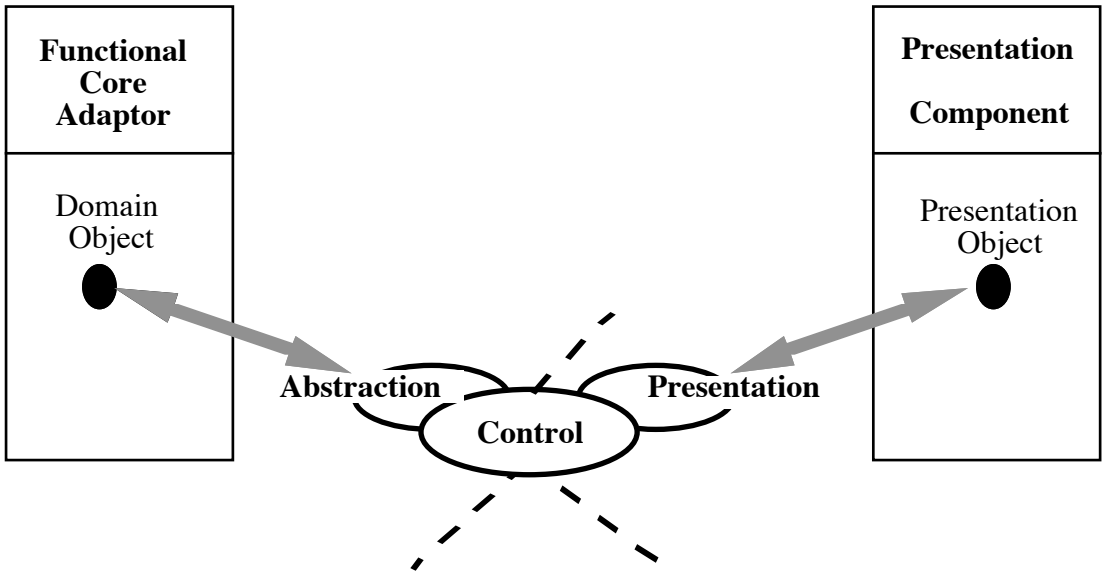

Figure 9: A PAC agent of the Dialogue Controller. Dashed lines represent possible relationships with other agents. Dimmed arrows show the possible links with the surrounding components of the Dialogue Controller. [Nigay 91b]

This general description of the Dialogue Controller does not provide enough insight about how to define the agents for a particular interactive system. Heuristic rules have been devised by Nigay in Project Amodeus 1 and have been implemented in the form of an expert system, PAC-Expert [Nigay 91a]. PAC-Expert is able to automatically identify the agents as well as their role from the description of the user interface of the system. The rules are organized along three criteria: window existence, window content, and links between windows.

5.5. The Implementational Models and the MSM framework

The following discussion is structured according to the dimensions of the MSM framework.

5.5.1. Communication channels

The notion of communication channels is implicitly covered by the implementational models. In Seeheim, they are located in the Presentation component. In Arch and PAC-Amodeus, they are part of the Interaction Component.

5.5.2. Levels of Abstraction

Within implementational models, levels of abstraction are correlated with the granularity of their components. Figure 10 illustrates the relative ordering of Seeheim, Arch and PAC-Amodeus according to the component granularity.

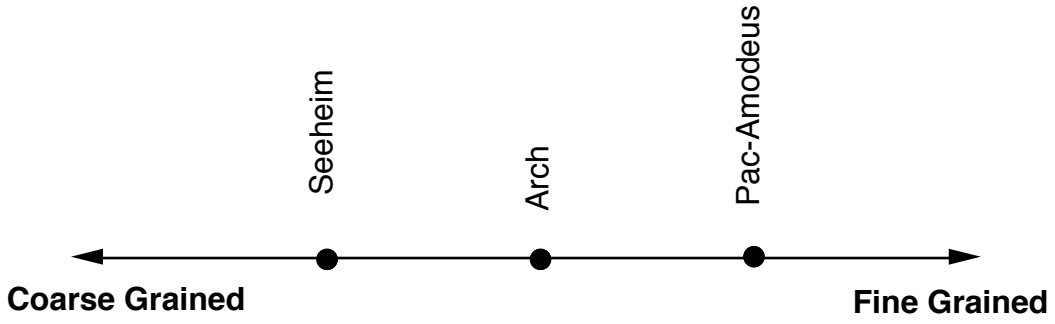

Figure 10: Ordering of the implementational models according to the granularity of their components.

Seeheim makes explicit four components from the low level Presentation to the high level functional core. Each component corresponds to a level of abstraction. Arch refines Seeheim on the concretization side by decomposing the Seeheim Presentation into two levels of abstraction: the Interaction Component and the Presentation Component. The Arch Presentation, which models a virtual Interaction layer, is an abstraction of the Interaction Component. Finally, PAC-Amodeus refines the Arch Dialog Component in terms of agents. This community of agents, which performs a sequence of abstraction and concretization operations, defines a set of levels of abstraction.

At the opposite of Seeheim, Arch and PAC-Amodeus make explicit the nature of the information exchanged between the components. This information successively takes the form of Interaction objects, Presentation objects, Conceptual objects, and Domain objects. Each one determines the interface between two consecutive levels of abstraction.

5.5.3. Context

Context is supported implicitly by the implementational models. One can view each level of abstraction as a context-based processing unit. Again, we have the intuition that this notion is important but we lack a clear definition to produce a sound discussion about this issue.

5.5.4. Fusion and Fission

Fusion and fission are not the main motivations of Seeheim and Arch. The Seeheim description however presents the Dialogue Control as the location for syntax analysis. By definition, a syntax analyser combines lexical items into syntactic units. Although not explicitly stated in the description of the Arch model, one can easily infer fusion and fission phenomena to transform Interaction objects into Presentation objects, then Presentation objects into Conceptual objects, up to Domain objects and vice versa.

Using PAC-Amodeus to develop Matis [Nigay et al. 93], we have identified three levels of fusion: lexical, syntactic and semantic that can be mapped to the three conceptual levels defined by Foley et al. [Foley et al. 84]. Lexical fusion corresponds to the Binding level which establishes the interface with the hardware primitives. Therefore lexical fusion is performed in the Interaction component. The syntactic and semantic fusions correspond respectively to the Sequencing and Functional levels. These fusions are thus handled by the component responsible for task-level sequencing: the dialogue controller.

Lexical fusion. Lexical fusion is performed in the Interaction Component. A typical example of lexical fusion may be found in the Macintosh where the shift key combined with a mouse

click allows multiple selections. Lexical fusion involves only temporal issues such as data synchronization.

Syntactic and semantic fusion. The Dialogue Controller is responsible for syntactic and semantic fusions. Syntactic fusion involves the combination of data to obtain a complete command. Semantic fusion combines results of commands to derive new results. For instance, in VoicePaint, the combination of the command “Draw line” with the command “Modify color” results in a two color line. (These two commands can be specified simultaneously.)

Syntactic and semantic fusion requires a uniform representation: the melting pot object. As shown in Figure 11, a melting pot object is a 2-D structure. The structural parts correspond to the structure of the commands that the Dialogue Controller is able to interpret. Events generated by user's actions are abstracted within the Presentation Component and mapped onto the structural parts inside the Dialogue Controller. These events may have different time-stamps. A command is complete when all of its structural parts are filled up by at least one piece of data. Multiple data for the same structural part may denote redundancy or reveal inconsistencies.

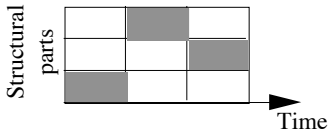

Figure 11: The melting pot object as a common representation for data fusion within the Dialogue Controller.

The non-sequential, hierarchical and distributed features of the multi-agent architecture adopted for the Dialogue Controller make it particularly well suited to perform fusion. Data is combined in parallel and incrementally along the levels of the hierarchy. The fusion mechanism is composed of a set of micro-fusions performed within each agent. The fusion process is based on two criteria: time (e.g., data belonging to the same temporal window) and the structure of the objects to be combined. Furthermore, an agent may add new data from its own state, to the fusion process.

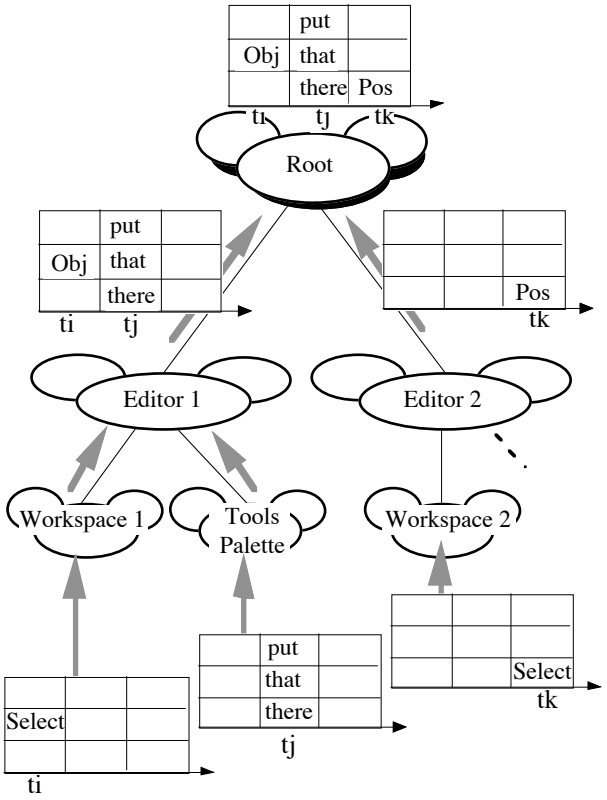

Figure 12: An example of different levels of fusion inside the hierarchy of PAC agents.

An example of fusion. Figure 12 illustrates a two-level fusion process for a graphics editor that supports speech and mouse gesture. In this example, the user says "put that there" and at the same time, uses the mouse to select the object to be moved and to indicate the destination in a distinct workspace. A workspace is a drawing area. As in most graphics editors, each workspace has a companion window, a palette that displays the graphics tools. By applying the heuristics rules described in [Niagy et al. 91a], one obtains the architecture shown in Figure 12.

At the bottom of the hierarchy, agents Workspace1 and Workspace2 interpret the events that occurred in the drawing areas. Similarly, ToolPalette agent is in charge of the events issued in the palette. Editor agents, such as Editor1 and Editor2, combine information from lower levels into higher abstractions. For the particular example, the three agents Workspace1, ToolPalette, and Workspace2 each receive a melting pot object from the Presentation Component. Each melting pot object corresponds to a user's actions. The agent Workspace1 translates the Select action into the selected graphical object Obj while in parallel, the agent Workspace2 translates the Select action into a position Pos. The cement agent, Editor1, then performs a first level of fusion by combining the "put that there" with the selected object. A second level of fusion is then performed by the Root agent to obtain the complete command to be sent to the functional core.

5.5.4. Parallelism

Seeheim and Arch do not support parallelism explicitly whereas PAC-Amodeus refines the Dialogue Component in terms of agents. However, one can inject parallelism at different levels of abstraction. Raw data is captured in the Interaction component by event handlers. There is one event handler per input device and the combination of multile event handlers under the control of a system process form a communication channel.

Concurrency is also supported in the Presentation Component which receives low level events (Interaction objects) from the Interaction Component and transforms them into more abstract interaction techniques. For example, a mouse click is transformed into the Select interaction technique. Finally, the multi-agent architecture of the PAC-Amodeus Dialogue Controller offers an interesting conceptual framework to support concurrency. Agents can process data (i.e., Presentation objects and Conceptual objects) in parallel.

5.6. Summary

In summary, if we refer to our simple classification scheme described in Section 2, Seeheim, Arch, Slinky and PAC-Amodeus are all driven by implementation issues: all of them introduce programming interfaces explicitly to augment portability and reusability. With regard to granularity, Figure 10 shows their relative ordering from the less refined, Seeheim, to the most refined, PAC-Amodeus. As far as the MSM criteria are concerned, PAC-Amodeus, which explicitly captures the notion of agent, inherits all of the advantages of agent-based approaches.

6. Conclusion

This document is a reflexion about current practices in architecture modelling for interactive systems. We have organized the presentation of architecture models along the "nature" dimension of our simple classification scheme of architecture models: conceptual versus implementational. We have also checked how these models are able to satisfy the MSM framework.

From there, we need to study how conceptual and implementational models can be used to model and implement multimodal user interfaces. On the conceptual side, GIO and PAC are good candidates. On the implementation side, PAC-Amodeus has already been validated with practical experiences. However, fission and interleaving between input and output are not clearly mastered yet.

7. Bibliography

[Abowd 91]
G. Abowd, Formal Aspects of Human-Computer Interaction, PhD Thesis University of Oxford, YCS 161 (1991), University of York, Department of Computer Science, Heslington, York, Y01 5DD, UK, 1991.

[Abowd 92]
G. Abowd, J. Coutaz, L. Nigay, "Structuring the Space of Interactive System Properties", IFIP TC2/WG2.7 Working Conference on Engineering for Human Computer Interaction, Elsevier Publ., Ellivuori, Finland, August, 1992.

[Arch 92]
Arch, "A Metamodel for the Runtime Architecture of An Interactive System", The UIMS Developers Workshop, SIGCHI Bulletin, 24(1), ACM, January, 1992.

[Bass 91]
L. Bass and J. Coutaz, Developing Software for the User Interface, Addison Wesley Publ., 1991.

[Blattner 90]
Blattner M.M. and R.G. Dannenberg R.G. CHI'90 Workshop on multimedia and multimodal interface design. SIGCHI Bulletin 22, oct., 1990, pp. 54-58.

[Coutaz 87]
J. Coutaz, "PAC, an Implemention Model for Dialog Design", Proceedings of Interact'87, Stuttgart, September, 1987, pp. 431-436.

[Coutaz 91]
J. Coutaz and S. Balbo, "Applications: A Dimension Space for UIMS's", Proceedings of the Computer Human Interaction Conference, ACM, May 1991, pp. 27-32.

[Coutaz 93a]
J. Coutaz, L. Nigay, D. Salber, "A Software Design Space for Multi-sensory-motor Interactive Systems", The Amodeus Project, Esprit Basic Research 7040, Amodeus Project Document, SystemModelling/WP4, 1993.

[Coutaz 93b]
J. Coutaz, L. Nigay, D. Salber, J. Caelen, "The MSM Framework: a Design Space for Multi-sensory-motor Interactive Systems", Position Paper, Workshop on Multimedia, Multimodal User interfaces, Interchi'93, Amsterdam, 1993b.

[Demazeau 91]
Y. Demazeau and J-P Müller, "From Reactive to Intentional Agents", Decentralized Artificial Intelligence, vol. 2, Demazeau & Muller eds., North Holland, Amsterdam, 1991.

[Duce 91]
D. Duce, M.R. Gomes, F.R.A. Hopgood, J.L. Lee (eds), "User Interface Management and Design", Eurographics Seminars, Berlin: Springer-Verlag, 1991, pp. 36-49.

[Duke 92]
D. Duke, M. Harrison, "Abstract Models for Interaction Objects", The Amodeus Project, Esprit Basic Research 7040, Amodeus Project Document, SystemModelling/WP1, 1992.

[Duke 93]
D. Duke, M. Harrison, "Towards a Theory of Interactors", The Amodeus Project, Esprit Basic Research 7040, Amodeus Project Document, SystemModelling/WP6, 1993.

[Faconti 93]
G. Faconti, "Towards the Concept of Interactor", The Amodeus Project, Esprit Basic Research 7040, Amodeus Project Document, SystemModelling/WP, 1993.

[Foley 84]
J.D. Foley, A. Van Dam, The Fundamentals of Computer Graphics, Addison Wesley, 1984.

[Golberg 84]
A. Goldberg, Smalltalk-80: The Interactive Programming Environment, Addison-Wesley Publ., 1984.

[Green 85]
M.W. Green, The Design of Graphical Interfaces, PhD Thesis, Tech. Report CSRI-170, Computer Systems Research Institute, University of Toronto, Canada, M5S 1A1, April, 1985.

[Harrison 90]
M. Harrison and Thimbleby, Formal Methods in Human Computer Interaction, Cambridge University Press, 1990.

[Hayes-Roth 79]
B. Hayes-Roth and F. Hayes-Roth, "A Cognitive Model for Planning", Cognitive Science, 3, 1979, pp. 275-310.

[Krasner 88]
G.E. Krasner, S.T. Pope, "A Cookbook for using model-view-controller user interface paradigm in Smalltalk-80", Journal of Object Oriented Programming, August-September, 1988, pp. 26-49.

[Krueger 90]
M. W. Krueger, Artificial Reality II, Addison-Wesley Publ., 1990.

[McCall 77]
J. McCall, Factors in Software Quality, General Electric Eds, 1977.

[Moran 81]
T. Moran, "The Command Language Grammar: a representation for the user interface of interactive computer systems", International Journal of Man-Machine Studies, 15, 1981, pp. 3-50.

[Nigay 91a]
L. Nigay and Coutaz, Software design rules for multi-agent architectures, Amodeus BRA 3066 RP2/WP17, 1991.

[Nigay 91b]
L. Nigay and J. Coutaz, "Building user interfaces : Organizing software agents", proceedings of ESPRIT'91 conference, Bruxelles, November, 1991.

[Nigay 93]
L. Nigay, J. Coutaz, D. Salber, "Matis, a Multimodal Air Travel Information System", The Amodeus Project, Esprit Basic Research 7040, Amodeus Project Document, SystemModelling/WP10, 1993.

[OSF 89]
OSF, "OSF/Motif, Programmer's Reference Manual", Revision 1.0; Open Software Foundation, Eleven Cambridge Center, Cambridge, MA 02142, 1989.

[Paterno 93]
F. Paterno, “A Methodology to Design Interactive Systems based on Interactors”, The Amodeus Project, Esprit Basic Research 7040, Amodeus Project Document, SystemModelling/WP6, 1993.

[Pfaff 85]
G.E. Pfaff and co-workers, User Interface Management Systems, G.E. Pfaff ed., Eurographics Seminars, Springer Verlag, 1985.

[Thimbleby 90]
H.W. Thimbleby, User Interface Design, ACM Press, Addison Wesley, 1990.

[Valdez 89]
J. Valdez, “XVT, a Virtual Toolkit,” Byte ,14(3), 1989.