A Metaphor and a Conceptual Architecture for Software Development Environments

Izhar Shy
Richard Taylor
Leon Osterweil

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Technical Report 89-23

A Metaphor and a Conceptual Architecture for Software Development Environments

Izhar Shy
Richard Taylor
Leon Osterweil

Department of Information and Computer Science
University of California, Irvine[1] 92717

14 July 1989

[1]This material is based upon work supported by the National Science Foundation under Award No.s CCR-8704311 and CCR-8996102-01, both with cooperation from the Defense Advanced Research Projects Agency, by the National Science Foundation under Award No.s CCR-8451421 and CCR-8521398, Hughes Aircraft (PYI program), and TRW (PYI program).

Contents

List of Figures

Abstract

A conceptual architecture for software development environments (SDEs) is presented in terms of a new metaphor drawn from business enterprises. A metaphor is employed as the architecture is complex, requiring understanding from several perspectives. The metaphor provides a rich set of familiar concepts that strongly aid in understanding the environment architecture and software production. The metaphor is applicable to individual programming environments, software development environments supporting teams of developers, and to large-scale software production as a whole.

The paper begins by considering three perspectives on SDEs, a function-based view, an objects-and-relations view, and a process-centered view. The process view, being the most encompassing, is held through the remainder of the paper. Three metaphors for organizing and explaining a process-centered environment are then examined, including the hierarchical contract model and the individual/family/city/state model. Next the *corporation* model is introduced and a detailed analogy is drawn between corporations and software development environments. Within the context of the corporation metaphor, three corporate organization schemes are reviewed and *federal decentralization* is argued to be most appropriate for an SDE. Relationships induced by such an organization are discussed and a mapping between the conceptual architecture and a possible implementation architecture is briefly discussed.

Keywords: models of software development; software development environments; process modeling; software factory; scalable, heterogeneous architectures; metaphors.

1 Introduction

The distinction between a system's conceptual architecture and its implementation architecture is between how the system is to be thought of by the system's users and how it is actually constructed. The conceptual architecture should reveal, to the end user, how the problem for which the system was constructed is addressed.

The presentation of a conceptual architecture for software development environments is a main contribution of this paper. This architecture is presented in terms of a new metaphor for software environments and indeed for software production as a whole. The metaphor is drawn from business enterprises. A metaphor is employed as the architecture is complex, requiring understanding from several perspectives. The metaphor provides a rich set of familiar concepts that strongly aid in understanding the environment architecture and software production.

The paper is organized as follows: a brief historical view of environment developments is presented first. This reveals our focus on process-centered environments. Three different metaphors for software environments are then considered, two of which are well-known, having previously appeared in the literature. The *corporation* metaphor is then introduced. Subsequent sections present the details of the metaphor and the corresponding components and aspects of software environments. Within the context of the corporation metaphor a particular kind of corporate organization, viz. federal decentralization, is then argued as particularly appropriate for organizing software environments.

While the metaphor and conceptual architecture are the contributions of the paper, the reader may not necessarily be convinced immediately that an achievable implementation architecture exists for environments organized according to the federal decentralization model. We therefore include a very brief section addressing this.

2 Views of Software Development Environments

Software development environment architects have taken several different approaches to supporting users in their tasks, including: a function view and an object and relations view. We consider these briefly, then describe our viewpoint, namely a process view.

2.1 Function View

In this view, environments are seen as consisting of relatively unstructured collections of functions invokable by "one" person.

UNIX (when considered as an environment) [KM81] and Interlisp [TM81] are examples of this approach. In UNIX the user is provided with low-level support

mechanisms that aid in building systems; function to function communication is achieved through byte streams. In Interlisp the functions are structured around an interpreter and a shared data structure for representing Lisp programs.

This view is deficient in a number of respects:

- the approach taken is low level — there is no particular organization of the active elements of the environment and there is little distinction between the conceptual architecture and the implementation architecture,
- only one person can interface with a tool at any one time, and most important
- the approach stresses the *way* tools operate and not *what* they produce, and certainly not *why* they do so.

2.2 Object and Relations View

Stoneman [BD80], and later Osterweil [Ost81], recognizing the limitations of the function model, proposed that, since the whole purpose of software development is to produce objects (code, documentation, design, etc.), an object-centered view of the environments should be adopted. In this view tools and project efforts are coordinated by access to a central *repository of information.* The set of objects captures the state of the system, and is manipulated by applying operations.

Such systems often store object histories (versions) as well as object derivations (structures indicating which objects have been produced from which other objects). It should be noted that, from this view, objects are considered passive entities, mostly responding to requests and not initiating activities on their own.

Odin [CO89] and Apollo's Domain Software Engineering Environment (DSEE) [LRPC84] are examples of environments integrated according to this view. In Odin, the environment is viewed as a collection of tools which are satellites around a large structured data repository. Similarly, DSEE provides a comprehensive set of data bases for coordinating the building and maintenance of software systems.

While a major improvement over the function view, some deficiencies are to be found.

- There is emphasis on *what* is being done while the details of *how* to generate an object are mostly overlooked. More to the point, the purpose of object generation (*why*) is still ignored.
- While software artifacts are organized in this view, there is typically no significant organization of the active elements in the environment. Moreover there is no clear way in which software environments *scale* from small object/tool bases to very large ones.

- The paradigm is well suited for writing a system which a single individual at a workstation can understand in its entirety, but is less suitable for multiple users.

2.3 Software Process View

In this approach environments are seen as entities to support a well-defined *process*. The process ideally captures the *purpose*, as pre- and post-conditions for a development activity, as well as the *plan* for orchestrating the development. As process-support entities, environments are thus concerned with the functions used and the objects built. In other words, this approach integrates the function view and the object view in the context of the purpose for this application or creation.

Another feature of this view is that it supports representation of how humans participate in the activities of the environment. Moreover, while the implementations of the two previous views have tended to presuppose that only one person interacted with environment facilities at a time, here multiple persons can participate in a way specified as part of the environment. The environment can support description of exactly how individuals interact, or do not interact, in cooperating to achieve the common goal of software development.

Arcadia [TBC*88] is an example of an environment being built using such a view, where the processes supported are programmed for individual project needs[1].

Some additional features that characterize this view are as follows.

- The approach invites a view in which machines and individuals operate concurrently and in co-operation with the goal of accomplishing software tasks, potentially yielding a finely structured organization of all the active elements in the environment.

- The paradigm is well suited for supporting the writing of systems in which several individuals operate at workstations, but where each individual is looking at only a part of the total system. A process can be constructed to coordinate the activities. Since it is expected that systems of the future will be more complex, embodying multiple users, machines and even subsystems, this approach will become increasingly attractive.

Additional advantageous features of this view will become apparent in the subsequent sections.

[1] One could also construct a process environment where the process supported was fixed. While such an environment might be useful in certain limited development settings, the limitations of such an approach are so obvious that further discussion of process environments is limited to those enabling extension, change, and development of new processes.

3 Process Organization in Software Development Environments

A useful approach to deciding the conceptual organization of a software process environment is to look for analogies in other domains where multiple users, activities, projects, and so forth form structured, coordinated aggregates. That is, we seek another domain where process is paramount. However, the selection of an analogy most useful in explicating the architecture of an SDE should also help determine the structure/organization of the environment. It should help determine the proper mechanisms that link the software development *problem* with its *solution*.

The following analogies are thus useful.

3.1 Hierarchical Contracts Model

In this model, proposed by Dowson [Dow86], each identified process in a software development activity is carried out in fulfillment of an agreed upon *contract*. The entities in the model play two roles.

- *Clients,* which define by means of a contract, the deliverables, acceptance criteria, resources, schedules, and reporting requirements to be followed.
- *Subcontractors,* which conduct activities as specified by clients.

This model has been implemented and is available as a commercial product named ISTAR.

While the model exhibits some clear strengths, some significant limitations can be identified.

In general the contracting structure allows too little interaction. Although it is possible for contracts to be, in turn, subdivided into additional subcontracts, the organization allowed by the model is restricted to a pure tree structure. Moreover sharing of data between contracted tasks is not allowed. Each contractor can proceed autonomously, recording all pertinent information in a local "contract database". No provision is made for sharing of such information between contractors, however. Formal channels of communications strictly follow the organization structure of a project. No lateral communications between contractors exist. This makes communication among sibling and cousin tasks expensive and ungainly.

3.2 Individual/Family/City/State Model

This model was proposed by Perry and Kaiser [PK88] at ICSE9 and views environments as consisting of policies, mechanisms, and structures. Policies are the rules imposed on the users, mechanisms are the functions (or tools), and the structures are the objects on which such tools operate. The model then uses a sociological

metaphor, classifying environments into four categories: individual, family, city, and state on the basis of the policy/mechanism/structure criteria.

- In the individual model, mechanisms (tools) and their construction are emphasized, UNIX and Interlisp once again being typical examples of such environments.
- In the family model structures (objects) and coordination between them are the main aspects. Odin, RCS [Tic82], SCCS [Roc75] and DSEE are examples for such environments.
- In the city model policies and cooperation mechanisms are characteristic. ISTAR and INFUSE [PK87] (an environment developed by Perry and Kaiser) are viewed as instances of such environments.
- The state model represents the the culmination of high level policies implemented on common mechanisms and structures. There are no instances of this model in any existing environments, according to Perry and Kaiser.

This classification of SDEs is similar to the view of SDEs proposed in this paper. The individual model matches our function view and the family model our object view. The city and state models do not adequately capture the notion of a software process environment, however.

The approach taken by the authors has some further limitations:

- Perry and Kaiser have not drawn from the analogy any details concerning how to build an environment. No refinement was proposed to expand on the abstractions; the major concern seems to be only the creation of a taxonomy for SDEs and not determining SDEs architectures.
- The individual/family/city/state structure implies a hierarchy: families contain several individuals, cities several families, and so forth. The analogy fails, however, when an attempt is made to apply it to environments. No hierarchies nor any formal relationships are found to exist between the various SDEs presented. For example, Unix is not a *member* of the RCS family, nor is Odin a family within INFUSE.
- The sociological metaphor chosen does not seem to provide any clues as how environments should be architecturally built, nor how they operate as far as policies/mechanisms/structures are concerned.

To summarize, the taxonomy used in the model deals mainly in scale and relationships within environments. It can be argued that the model proposed is not sufficiently rich to serve as a blueprint of how to build a highly integrated environment which is nonetheless extensible, scalable, and flexible.

3.3 Corporation Model

This new model draws similarly from a sociological metaphor; software development environments are seen as analogous to corporate organizations. We will show how this metaphor avoids the problems mentioned above and in fact presents useful guidance in the design of a SDE.

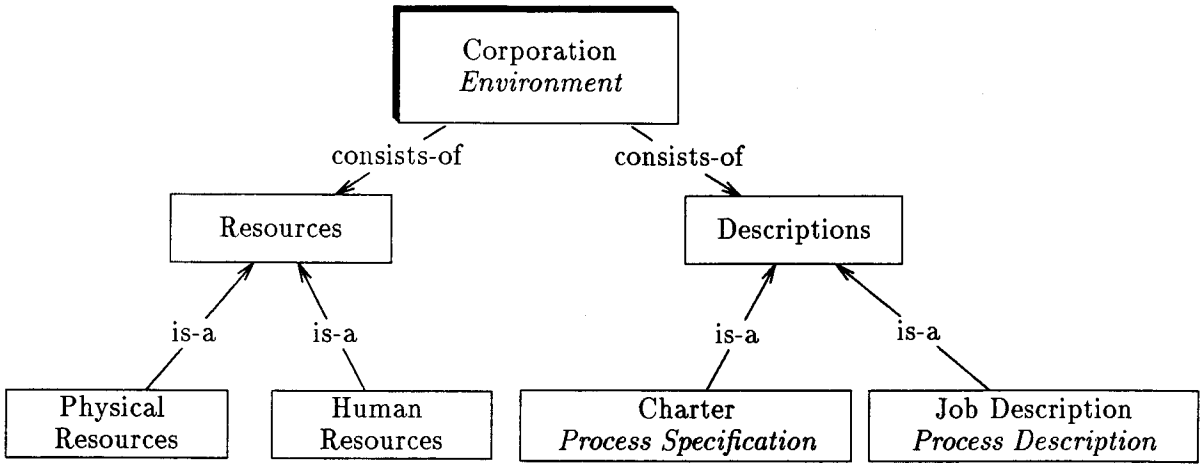
We can envision a corporation as consisting of four major components: Resources, Descriptions, Support Groups, and Active Agents. *Resources* in a business enterprise are of two types, physical and human. A corporation's *descriptions* include its charter which is a description of its goals, nature of its business, and products, and its corporate job description. *Support groups*, such as payroll or facilities maintenance, provide services to other components of the corporations. A corporation's *active agents* are its managers and associates. Managers are responsible for directing the activities of Associates who execute such activities as designated in their job description or statement-of-work. Resources are provided by managers to associates in order to enable performance of the work assigned to them.

Similarly we argue that in process-oriented software development environments there are also four major components: Resources, Descriptions, Infrastructure, and Active Agents. Resources are people, objects, processors, and so forth. Descriptions are process specifications which indicate the purpose and objectives of an environment and process descriptions which are vehicles for expression, with the aid of programming techniques, of how the process specification is to be achieved. Environment infrastructure consists of all mechanisms necessary for automated interpretation of process programs. An environment's active agents are its software processes and software tools. Software processes represent execution of process programs, while software tools represent execution of tool descriptions [2].

It can therefore be seen that while the purpose of a corporation is to "support its owners" in some business activity (economic performance), the purpose of an environment is to "support its users" in software activities.

For abbreviation purposes we will use the term "agent" as a substitute for managers and associates in the corporation model and software processes and software tools in SDEs.

[2] As will be seen later the distinction between software process and tool is typically one of perspective, in the same way that to a president a vice-president is an associate, while to the vice-president's staff the vice president is a manager.



Legend:
Roman font = Corporation model
Italic font = Environment model
If box contains only Roman font, it applies to both models.

Figure 1: **Basic components of corporations and environments**

4 The Corporation Model

4.1 Description

4.1.1 Initial Components

Our corporation model portrays a corporation as consisting initially of two basic components: Resources and Descriptions. Figure 1 presents a view of corporations and environments consisting of these components.

Resources Economic activity requires two kinds of resources: physical resources such as land, buildings, offices, supplies and capital; and human resources — people. In order to operate successfully, a business enterprise must be able to attract both physical and human resources and put them in productive use.

In environments, software development activity requires physical resources such as objects, processors, and communication channels, as well as human resources that can perform creative development activities.

Charters/Job Descriptions Corporations are associations formed in order to carry on some commercial or industrial undertaking, and they provide the machinery

to implement projects and create products. Since corporations are "goal oriented" and they operate in most cases with the intent of generating profit, they must be provided with a charter which defines *inter alia*: its name, purpose, place of business, nature of business, names of incorporators, products, etc. In other words, the charter identifies the "goal and purpose" of the corporation.

Specifying the goals and purpose of an organization (the "what"), is not sufficient to encompass the description of its operation. It is also necessary to provide it with specifications of the jobs that have to be performed by it, i.e. how various jobs are to be carried out. Such specifications are available in the form of job descriptions, corporate "standard operating procedures", work breakdown structures, task orders, and the like.

Both charters and job descriptions are static, inert entities essential for the instantiation and activation of any business organization.

In the SDEs domain the *process specification* corresponds to the charter in the corporation domain. It identifies the the "goal and purpose" of software activities but does not specify the way such activities are to be carried out. The *process description* is roughly equivalent to the detailed work breakdown structures in the corporation domain but process descriptions are far more formal and specific [Ost87]. Process descriptions may also be viewed as *process programs.*

4.1.2 Active Agents

The initial components of the corporation — resources and descriptions — are unable by themselves to perform, i.e. to execute any work on behalf of the corporation. The actual work is performed by *active agents* created from the basic components. Figure 2 shows how such creation is taken place: physical and human resources are *assigned* to job descriptions in keeping with the charter thus creating active agents. In corporations such agents are *managers* and *associates.*

Similarly, in the SDE domain, resources such as people and processors machines are *bound* to process programs in keeping with the process specification creating active agents — *software processes* and *software tools.* The actual binding operation is performed by another active agent, and is discussed in section 4.1.3.

Managers/Software Processes Managers create a productive entity containing more than the sum of resources put into it and, at the same time, harmonize in their decisions and actions the requirements of immediate and long-range future. They are *active agents* provided with *goals* by the corporation and their superiors. In the SDE domain, managers have their analogue in *software processes.* The goals in software projects are the creation and maintenance of software products and such goals are achieved by execution of process programs.

Managers perform the following functions:

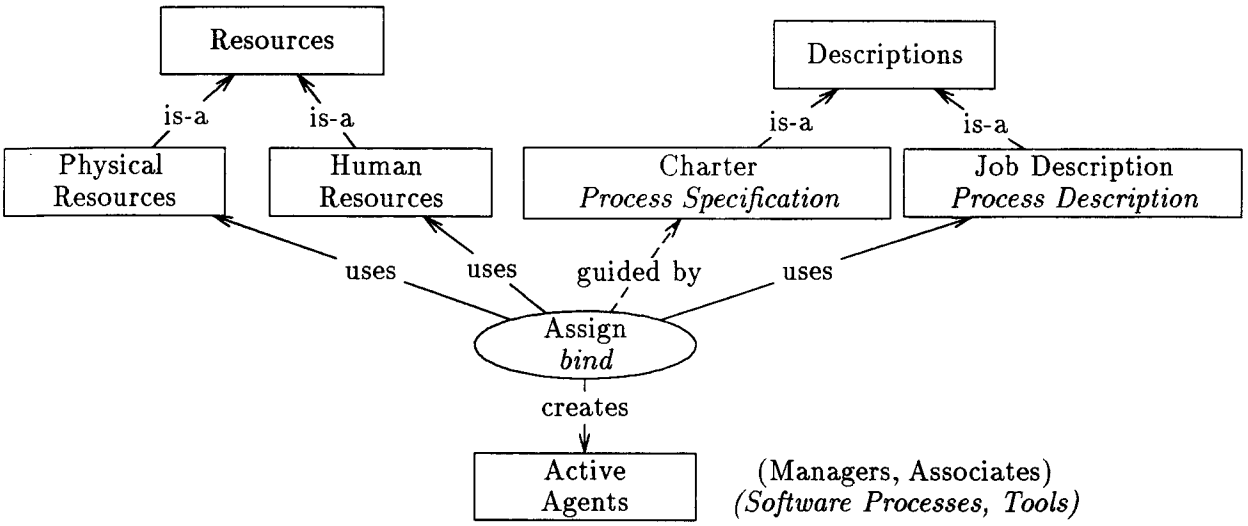


Figure 2: Creation of active agents

- *Set Objectives* - From goals provided to them by their superiors, they determine the objectives and the activities required to reach these objectives.

  In an SDE the goals are provided in the form of a project specification which lists requirements to be fulfilled in order to accomplish the project objectives.

- *Organization* - Managers analyze the activities needed, separate them into manageable tasks, group the jobs into an organization structure and assign personnel to perform these tasks. To perform required tasks, managers may need to provide the subordinates with other resources.

  In the SDE domain, project organization is accomplished by developing and enacting a *process program,* which is the description of activities necessary to be completed in order that the project objectives are met. Software processes, by executions of such process programs, similarly analyze the activities needed, separate them into tasks that can be performed by software tools and/or other subordinate software processes that are executed by people. Software process descriptions (process programs) are analogous to highly detailed management job descriptions, in that they are inert entities that come to life (become active) as soon as resources are attached (bound) to them. Like managers, once software processes receive resources and a "go-ahead" they become active agents.

- *Communication* - Managers consolidate subordinates into a successful team, by establishing and fostering the communications between themselves and their subordinates, between themselves and their superiors, between them-

selves and their colleagues, and between their subordinates. Similarly, software processes may establish and control communication links upwards, downwards and sideways.

- *Process Control* - Managers are also responsible to monitor their subordinates in order to ensure that proper project performance is achieved. They must be able to detect any deviations from the project specification and schedule and take appropriate measures to correct them. Similarly, software processes must be able to monitor and control the execution of subordinate processes and tools and to make adjustments or modifications in such subordinates that fail to perform properly.

- *Measurement* - Successful managers must establish criteria and yardsticks by which to analyze, appraise, and interpret the performance of their subordinates. Similarly, software processes must be provided with means for measurement of their subordinate software processes and tools[3] .

Associates/Software Tools Associates execute work designated by managers. As such, similarly to managers, they are "active agents". They are created by binding job descriptions or work breakdown structures to people.

Associates are robust in that they have:

- *autonomy* i.e. the ability to make decisions,

- *internal structure* i.e. the ability to subdivide their tasks thus becoming managers themselves,

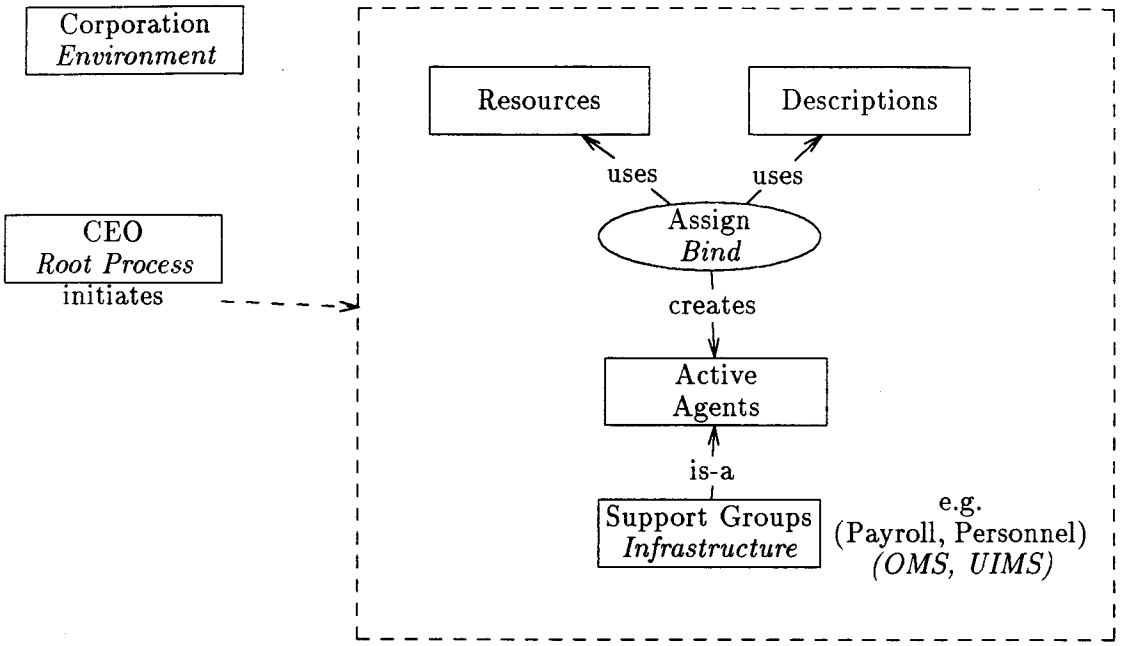
- *organizational ability* i.e. the ability to determine their interactions with other associates within same bounds.

In the SDE domain, associates find their corresponding entity in *software tools* and software processes executed by people. The robustness of software tools is evident in their ability to become software processes and recursively spawn additional software processes and tools.

4.1.3 Infrastructure Creation

In corporations the board of directors selects the Chief Executive Officer (*CEO*); in SDEs the boot operation creates the *Root Process*. These initial active agents are

[3] If we wanted to press the analogy even further, we could add:
Subordinate Development - Managers are responsible for helping their subordinates grow and improve their skills through training and personal example. Similarly some software processes may be allowed to modify other process programs to meet new objectives.



OMS – object management system
UIMS – user interface management system

Figure 3: Infrastructure creation

necessary to get the corporation/environment going. Typically they are immediatedly responsible for the creation of the infrastructure which provides support for the entire corporation/environment.

Figure 3 is intended to present such infrastructure creation: The CEO initiates the binding of resources to descriptions thus creating the *support groups* such as personnel, payroll, and legal. Similarly, the Root process initiates the binding of resources to descriptions creating the *environment infrastructure.*

Support Groups/Environment Infrastructure The support groups in a corporation are not direct product producing entities. Their function is to supply services to other entities in the organization. As an example, in almost all corporations, the "payroll entity" computes the wages for corporation employees, subtracts the appropriate deductions, disburses the net pay, and maintains records as required by various government and other agencies. In our view of environments, the analogs of support groups in the SDE domain are all the mechanisms necessary for the automated interpretation of process programs [TBC*88]. Figure 4, taken from that paper presents one opinion of what constitutes an appropriate infrastructure for an extensible process-centered environment. It consists of:

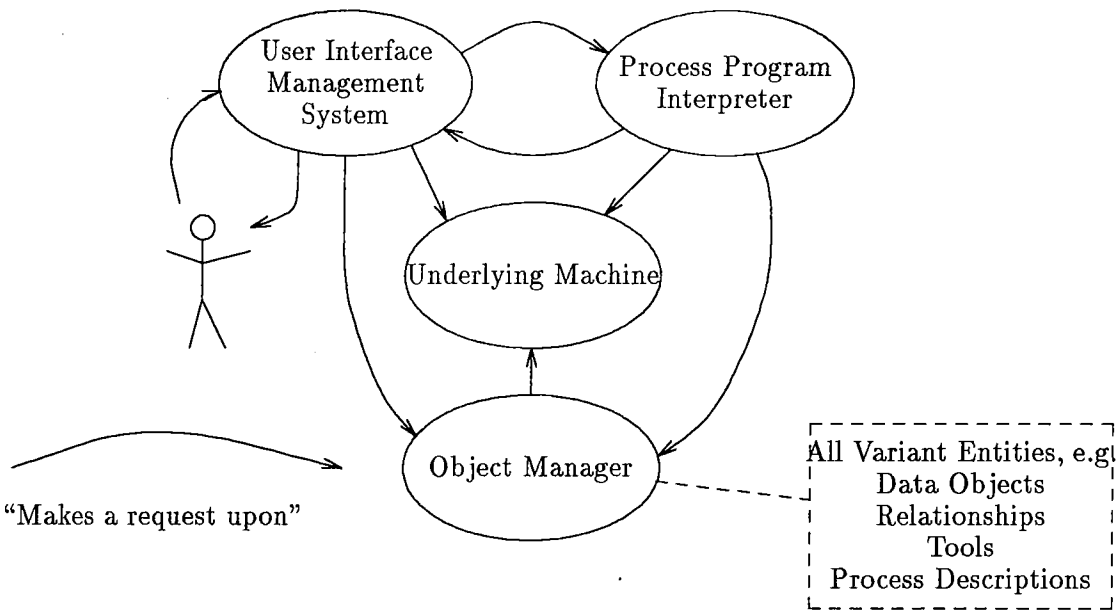


Figure 4: Infrastructure components and variant entities

- A process programming language system (PPLS) including a process program language and a system enabling the interpretation of programs written in the language.
- A object management system (OMS), which provides the facilities for managing persistent typed objects.
- A user interface management system (UIMS), which provides the human user with access to the functions supported by the environment[4].

4.1.4 Corporate/Environment Organization

In order to become active enterprises, in addition to the infrastructure, corporations need to create the proper organization structure by instantiating its operating units and providing them with proper resources.

Figure 5 shows how an agent within a *corporate unit* at one level can *instantiate* a unit at an immediately lower level by providing the new unit with its charter and job, as well as *granting access* to resources to be used by the new unit. The

[4]The "underlying machine" which encapsulates functions provided by a Virtual Operating System (VOS) finds its corporation model equivalent in those individuals or groups chartered with providing all foundational functions in order to ensure the proper operation of the environment infrastructure. Electricity, water, and similar other basic foundational services are examples in the corporation model.

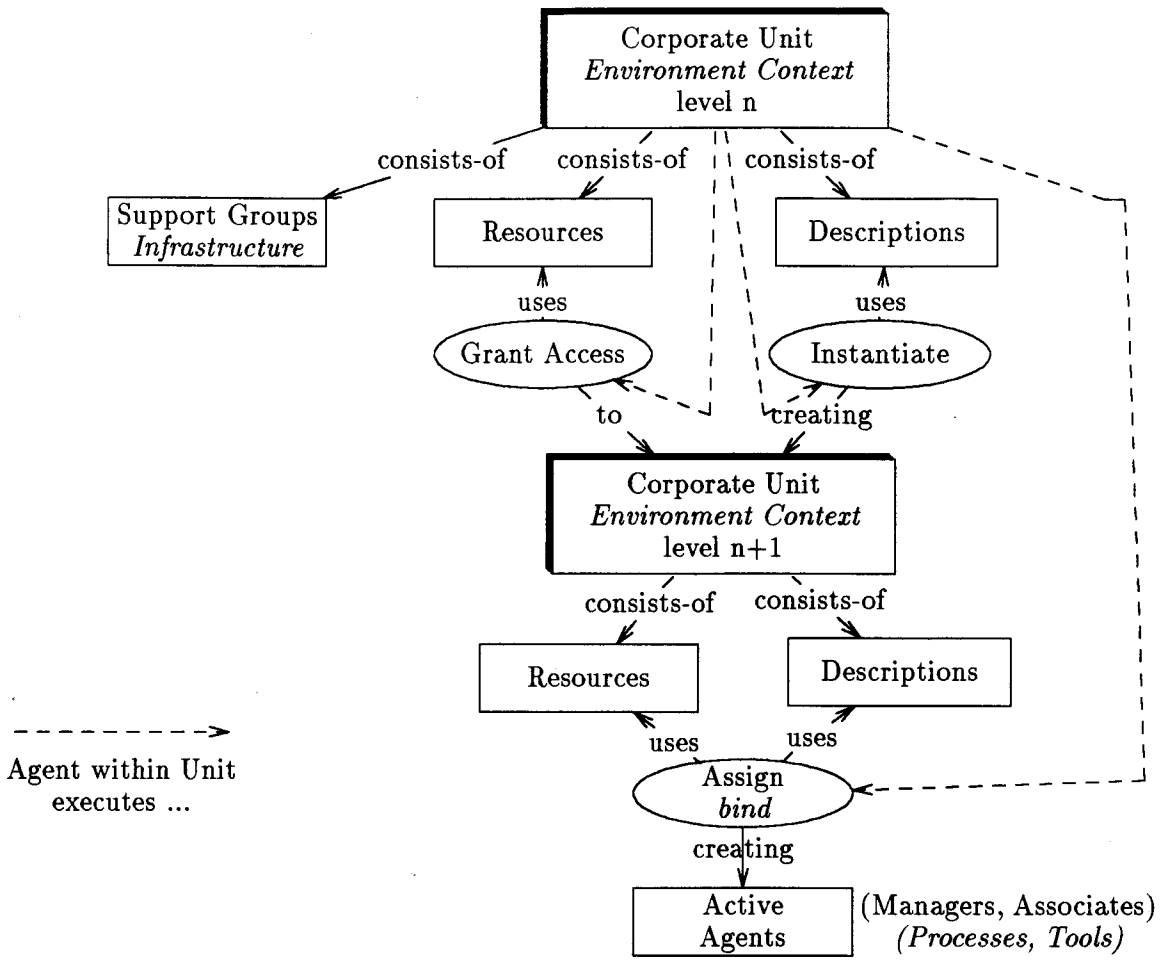


Figure 5: **Introducing hierarchy into corporations/environments**

agent within the original unit can then bind resources to descriptions creating active agents in the new unit.

In environments, the equivalent entity for the unit is the *environment context*, or just "context", and the figure shows the instantiation, resource access-granting, and binding required to create a lower level context.

Figure 6 presents a view of a corporate/environment unit after active agents have been bound and activated. Support groups/infrastructure are also shown.

It should be noted that the acts of instantiation, resource access-granting, and binding can potentially be performed by units at any level and thus present the mechanisms sufficient for the creation and activation of an entire organization, regardless of size. Similarly in SDEs it supports the model of small single user environments, through team projects, and up to very large project development

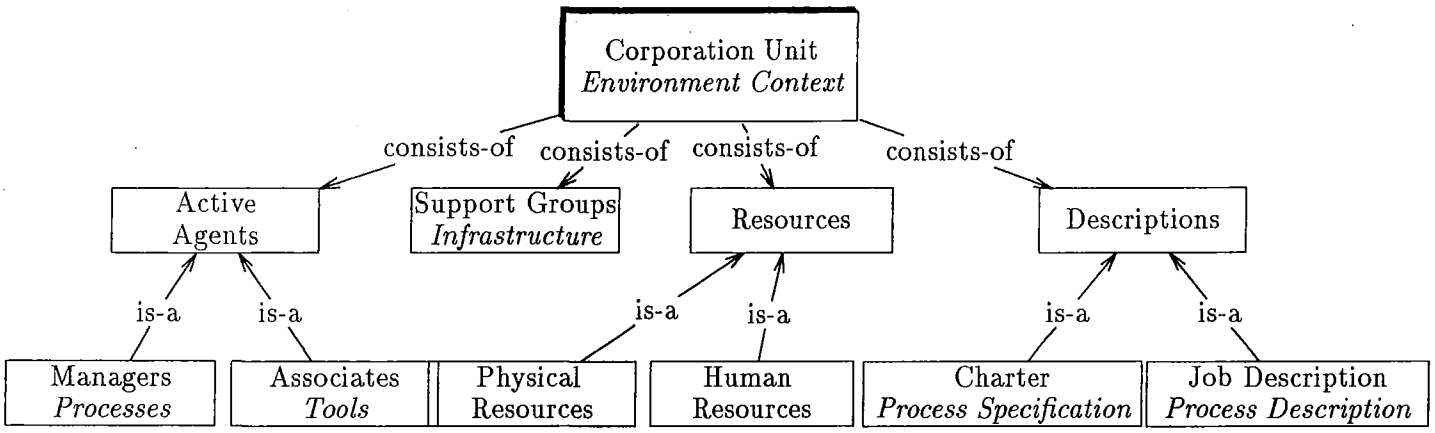


Figure 6: Components of a corporation unit/environment context

activities.

4.2 Organizations Induced by Relationships between Agents

No connection has yet been established between agents and the support groups higher up in the hierarchy of units. The agents in a corporation do not exist in a vacuum; they are interrelated to each other and to the corporation's support groups in order to achieve their goals. Depending upon the level of autonomy allowed the agents in a corporation it is useful to identify the following organizational structures.

4.2.1 Corporate Autocracy

In such a structure, agents utilize only global support services provided by the corporate headquarters for all mechanisms needed in carrying out their functions. The control on the structure of the organization is extremely tight and agents are not given any freedom in selecting the provider of services they need. For example, all divisions in an organization are directed, by policy, to utilize corporate services for payroll, computer services, legal, etc. They are specifically forbidden to provide such services within their own unit. Communication between agents may be proscribed to be strictly upwards and downwards in the hierarchy. The analogy in environments is to software processes and tools that use only global services, such as the OMS, UIMS, and PPLS given the example description of the components of an environment infrastructure in section 4.1.3.

4.2.2 Radical Decentralization

Here the corporation has no global support groups and all needed functions are performed by the agents themselves. The control on the structure of the organization

is loose; each entity has independence to conduct business as it sees fit. Communication between agents is non-existent and unnecessary since all functions are performed within the agent itself. A corporation example is any corporate holding-company where there are no restrictions on the local companies except that they are required to show profit. The local companies are free to construct their local support services which they can use at will.

In an analogous environment all OMS, UIMS and PPLS functions are performed entirely within the separate processes themselves. There is no global environment infrastructure and communication between active agents is non-existent.

4.2.3 Federal Decentralization

Here the corporation has global support groups but active agents also have some internal facilities that provide services. The control under this scheme is "flexible" allowing agents to determine how much central support to use. As a clarifying example we can view the structure and policies of the McDonalds corporation. Franchisees are allowed and encouraged to use service facilities as provided by the franchising headquarters, but they are also given some freedom to acquire such services at the local level, as long as the company image and standards are preserved. In environments this is equivalent to providing global environment infrastructure capabilities but at the same time allowing processes to include local internal versions of services such as an object manager. Just as relations between the agents of a federal decentralization organization must be negotiated and mediated, it is also true that relations between processes in a federated process environment must be mediated. Consider for example the requirement that all communications between agents within Quebec corporations be made in French; however agents that are more comfortable communicating in English may do so by agreement. In environments such agreements could, for example, be mediated and achieved by a Unified Type System (UTS) service [WWRT89]. In an environment two agents with different object managers may have different type systems or models inhibiting communication. The UTS service provides these agents with the means to communicate in spite of the different type models used.

The federal decentralization model seems to us to offer flexibility and robustness and seems to serve as a good basis for environments because:

- The level of usage of global support services is not rigid but it is determined by agents at all level in the organization. In some instances such determination can be dictated from above, in other it can be negotiated between agents, and in still others it can be arbitrarily determined by the agents themselves. A wide spectrum of possibilities are thus available.

- Mechanisms for communication between agents (upwards, downwards and sideways) are provided and maintained under the control of the agents them-

selves. The communication however, is established under guidelines that are determined by the policy (process specification) of the organization.

- Flexibility in the structure allows for diversity of services. For example, organizations are able to experiment with different payroll systems concurrently in several units, without any disturbance to the global payroll service provided to remaining units.
- Organizations become extensible with minimum disruption since integration of services can be implemented gradually.
- Organizations can grow easily because the support services can be used from many levels within the organization.
- The model allows for foreign tool importation; it is analogous to having a corporation acquire a new corporation, allowing it to operate as an semi-independent unit.

5 Relations among Contexts and Agents

Agents exist in both the corporate model and SDEs; they interact with each other in a variety of ways according to relationships established between them. We now describe various of the key relationships needed in a federal decentralization process environment.

5.1 Instantiator/De-instantiator of

In the corporate model, to instantiate or create a new unit is to generate a new entity that is identifiable by name. From this moment on, the new unit is recognized as "existing" as a member in the community of units including all the machinery required to interface with the support staff within the corporation. This involves creation of a new unit in a corporation, providing it with its charter and job descriptions, and naming its manager. It should be noted here that the naming of a manager in no way implies that such a unit has been activated and made operational. The instantiation action is best illustrated in figure 5.

In SDEs agents within contexts instantiate other contexts; the new contexts becoming members of the community of contexts and containing machinery to interface with the environment infrastructure (PPLS, OMS and UIMS) and other VOS services. It should also be noted that neither transfer of resources from the instantiator, nor binding of any resources in the instantiated context has taken place. The instantiation of a new context does not activate it either (see below).

We hypothesize that context instantiation is a fundamental activity in an environment and each context can have only one instantiator. Therefore the "instantiator of" link must establish a context instantiation in a tree hierarchy. Only an instantiator is allowed to de-instantiate its instantiated children. In order to provide robustness, the recursive de-instantiation of contexts should be one of the design objectives in SDE architecture.

In the corporate model transfer of entire units from one manager to another are possible. Similarly, in SDEs the position of a context in the instantiation hierarchy is not fixed; an instantiator is allowed to move any of its dependents within its subhierarchy.

5.2 Grants/Denies Access to Resources

Resources in corporations reside within corporate units and are under the ownership and control of the unit manager. Therefore, a manager in a corporation at any one time possesses one or more resources required in order to accomplish his mission and, at his discretion, may grant or deny access to such resources to other managers, including subordinate managers. Since instantiation does not imply transfer of access rights to the newly created organization, a relationship is required in order to provide managers and associates in units with such access. The relationship of granting rights to resources does not have to follow the instantiation hierarchy; it is entirely separate. In corporations granting access to resources implies physical transfer of such resources to the recipient.

In the SDE domain, software contexts own resources and may grant or deny access to such resources to other contexts. Here as well the granting of access rights need not follow the instantiation hierarchy. Some systems (UNIX for one) automatically grant access rights to descendents upon creation; however this burdens some processes with access to resources for which there is no need or use.

5.3 Binding/Unbinding Resources

Managers in corporations assign (bind) people (human resources) to job descriptions. Such binding evolves them into active managers and associates.

Similarly, in the SDE domain, a software process binds resources to descriptions. Binding of computer processors or people to process programs evolves them into active software processes. It should be noted that a process can be granted access rights to resources owned by various other processes while this binding action can be initiated by a still different process. This allows for substantial flexibility in resource control in the environment.

5.4 Corresponding with

In the corporate models there are formal and informal lines of communications between managers and their superiors, colleagues, and associates. The formal lines are established by the corporate organization charts and by a set of rules and procedures, while the informal ones are established through day by day operation of the corporation. In general, the formal lines of communication within an establishment follow the management control structure, i.e. managers communicate only with their immediate superiors and their subordinates. This arrangement is, of course, suboptimal and additional informal channels of communications, not necessarily implying any authority or control, are established routinely.

In the SDE domain, the formal lines of communications are represented as communications along the instantiator relationships. Analogously to the informal communications analogue, a process may or may not grant another process the right to send messages to it and the expectation of receiving responses from it. It can be seen that there is a close similarity between the "own resource" and the fact that a process owns the receiving communication mechanisms. As such only the process which is the owner of these mechanisms can determine which other processes are allowed to communicate with it.

The last three relationships (grant access, binding, and corresponding) handle resources. The agent that has the authority to grant access, bind or establish correspondence is the only agent allowed to respectively deny access, unbind and disallow communications to the receiving agent. It should be realized that such a requirement presents difficulties in housekeeping but it is the only method by which sufficient control can be exercized over relationships between agents and contexts.

5.5 Communication issues

The "corresponding with" relationship serves as the basis for establishment of communication between processes.

The information that can potentially be communicated is as follows:

- "objects" as defined by the object management component.
- resources/capabilities belonging to the environment (e.g. identification of appropriate user interface servers, resources belonging to the underlying abstract machine).

This is because entities either belong within the domain that the environment is dealing with, or are part of the implementation of the environment.

To ensure proper communication linkages between environment contexts and agents it is necessary to observe the following rules:

- A creator task has the inherent capability of communicating objects and resources to its createes (dependents), so that the objects and resources needed by the dependent to do its work can be transmitted,
- A created task has the inherent capability of communication with its creator, to return developed products and, e.g., request additional resources,
- A process is the sole owner of its resources and communication capabilities. As such only a process can assign the capability of communication with it to others. In other words the process determines from whom and to whom it is willing to accept and send messages.
- capabilities may be taken away by creators.

The first two capabilities mirror the kind of formal communication links established in the corporate model. The third capability reflects the informal communication paths that get established within an organization.

6 Mapping from a Conceptual Architecture to an Implementation

One aspect of an implementation architecture is provision of components to manage the relations described in section 5. Thus functions to manage resources, establish communications, and oversee creation of new contexts are required. Components to support the other actions of agents, such as listed in 4.1.2 are also required. Included here e.g. is functionality to enable measurement of one associate by its manager. These primitives must be able to maintain consistency among the relations and enforce the policy decisions contained in a context's descriptions.

Such services can be provided reasonably. Our confidence here is based on recognizing the similarity of the functional primitives identified through this top-down design exercise with the functions identified, e.g. in [TBC*88] as components of an environment infrastructure. One could argue that an object management system or a user interface management system are substantial technologies that require the primitives resulting from our top-down design. Since [TBC*88] identified such components in a bottom-up fashion — drawing from experience with many implemented environments, we foresee no barriers to an implementation.

Consideration of some general characteristics of the conceptual architecture also identifies some desirable properties of an implementation substrate. Since corporations are highly concurrent entities it is clear that environments should be as well. Implementation primitives for managing concurrency are therefore required.

The hierarchical decomposition of a corporation, considered with the limited heterogeneity of the federal decentralization model, argues for mechanisms to support component implementation that follow the client/server model. That would enable effective use of a support service within one context by agents in another. While other desirable properties could be enumerated, we think it is clear that an effective implementation of the conceptual architecture presented in this paper could fully utilize the novel characteristics of modern distributed operating systems.

7 Summary and Conclusion

Current activities in modeling and developing software development environments have produced a number of proposals for conceptual architectures focusing on various metaphors. This paper has presented a view that is useful in deriving a sound conceptual architecture from which several implementation architectures are possible.

It is our contention that the *software process view* is the most encompassing and comprehensive and that it is the view that be taken in the conceptual architecture of SDEs in the future.

Another issue as presented here is the development of a conceptual architecture based upon sociological metaphors. Here the proposed corporation model seems to provide useful insights into the structure and organization of an SDE that can serve as a foundation for its conceptual architecture.

It should be noted that the proposed model differs in some key ways from the "software factory" model [Mat87]. The factory model tends to suggest a structure and organization where most of the operations are geared to the generation of specific products, and such operations are mostly mechanical, automatable, and, surely, easily organized. As such, little flexibility or creativity is possible or expected. On the other hand, our corporation model enables much more prominence to be placed on the role of creative individuals. Furthermore modern corporations are highly flexible and allow for more complex structures in their organization.

Lastly, we note that Conway [Con68] postulated that "there is a very close relationship between the structure of a system and the structure of the organization which designed it". The statement was meant in a derogatory way, i.e. that systems tend to be complex, cumbersome, and difficult to use and understand because their structure is homomorphic to the producing organization. Our thesis is different: modern corporations exhibit remarkable similarities across diverse product ventures and even different cultures. The common character and success of corporations is, we believe, due to their ability to effectively manage complex development activities in which change to products, change to the processes that control product creation, and change to the organizational structure itself are common. Since complexity and change characterize software development we can learn and apply principles from

this other domain.

Acknowledgements

We gratefully acknowledge the contributions made by members of the Arcadia consortium in review, criticism and encouragement provided to the authors throughout the development of this paper. In particular we wish to thank Barry Boehm for pointing out the advantages of the corporation metaphor over the software factory metaphor as well as some of the control functions performed by managers.

References

[BD80] John N. Buxton and Larry E. Druffel. Rationale for Stoneman. In *Fourth International Computer Software and Applications Conference*, pages 66–72, Chicago, IL, October 1980.

[CO89] Geoffrey M. Clemm and Leon J. Osterweil. A mechanism for environment integration. *ACM Transactions on Programming Languages and Systems*, 1989. To appear.

[Con68] Melvin E. Conway. How do Committees Invent. *DATAMATION*, 14(4):28–31, April 1968.

[Dow86] Mark Dowson. ISTAR – an integrated project support environment. In *Proceedings of the Second ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 27–33, Palo Alto, California, December 1986. Appeared as *SIGPLAN Notices 22*(1), January 1987.

[KM81] Brian W. Kernighan and John R. Mashey. The UNIX Programming Environment. *IEEE Computer*, 14(4):12–24, April 1981.

[LRPC84] David B. Leblang and Jr. Robert P. Chase. Computer-aided software engineering in a distributed workstation environment. *ACM Sigplan Notices*, 19(5):104–112, May 1984. (Proceedings of the First ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments).

[Mat87] Yoshihiro Matsumoto. A Software Factory: An Overall Approach to Software Production. In Peter Freeman, editor, *Tutorial: Software Reusability*, pages 155–178, IEEE Computer Society Press, 1987.

[Ost81] Leon J. Osterweil. Software environment research: Directions for the next five years. *IEEE Computer*, 14(4):35–43, 1981.

[Ost87] L. J. Osterweil. Software processes are software too. In *Proceedings of the Ninth International Conference on Software Engineering*, pages 2–13, Monterey, CA, March 1987.

[PK87] Dewayne E. Perry and Gail E. Kaiser. Infuse: A Tool for Automatically Managing and Coordinating Source Changes in Large Systems. In *ACM Fifteenth Annual Computer Science Conference*, pages 292–299, St. Louis, MO, February 1987.

[PK88] Dewayne E. Perry and Gail E. Kaiser. Models of Software Development Environments. *IEEE Transactions on Software Engineering*, 10(4):60–68, April 1988.

[Roc75] M. J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, 1(4):364–370, December 1975.

[TBC*88] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michal Young. Foundations for the Arcadia environment architecture. In *Proceedings of ACM SIGSOFT '88: Third Symposium on Software Development Environments*, pages 1–13, Boston, November 1988. Appeared as *Sigplan Notices 24*(2) and *Software Engineering Notes 13*(5).

[Tic82] Walter F. Tichy. Design, implementation, and evaluation of a revision control system. In *Proceedings of the Sixth International Conference on Software Engineering*, pages 58–67, Tokyo, Japan, September 1982.

[TM81] W. Teitelman and L. Masinter. The Interlisp programming environment. *IEEE Computer*, 14(4):25–33, April 1981.

[WWRT89] Jack C. Wileden, Alexander L. Wolf, William R. Rosenblatt, and Peri L. Tarr. *UTM-0: Initial Proposal for a Unified Type Model for Arcadia Environments.* Arcadia Technical Report UM–89–01, University of Massachusetts, Amherst, 1989.