Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Scherer

A Distributed Framework for Integrated Software Process and Deployment Support

TIK-Report
No. 52, July 1998

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

D. Scherer
A Distributed Framework for Integrated Software Process
and Deployment Support
July 1998
TIK-Report No. 52

Computer Engineering and Networks Laboratory,
Swiss Federal Institute of Technology (ETH) Zurich

Institut für Technische Informatik und Kommunikationsnetze,
Eidgenössische Technische Hochschule Zürich

Gloriastrasse 35, ETH-Zentrum, CH-8092 Zürich, Switzerland

ABSTRACT

Software development and deployment are traditionally separate non-interoperating tasks, yet they concern much of the same information. We propose a modeling language for software development processes that incorporates the software artifacts being developed in the process as well as their product structure information including evolution, such as dependency, origin, version, and configuration information which is required for deployment. Based on this, we present a distributed framework, implemented in a prototype system, consisting of a process support system as a common process-interoperable base for both developers and users, and separate enhancements for development and deployment. For developers, it is extended by a process definition and enactment engine as well as process-specific tools to make up a software engineering environment, allowing distributed development, configuration and release of software in globally linked processes. For users, it is extended by a deployment and runtime system that allows retrieval, installation and invocation of software, and manages user-site process bases. Developers publish parts of their development processes, and users copy linked parts of different developers' processes containing software artifacts to their local process bases. Deployed software retains its process-awareness, allowing configurations to be checked for compatibility.

Keywords

distributed framework, software development process, software deployment, process interoperability, distributed objects

1 INTRODUCTION

The Internet's remarkable growth and the emergence of distributed component technologies influence both software development and software deployment and demand their traditional requirements to be revised. Software is increasingly being developed as a "system of systems" by a heterogeneous group of organizations, in virtual organizations [2] or independently, requiring large and complex dependency graphs of software components to be managed. Likewise, software systems are becoming less self-contained, with numerous components from different sources required to make up an application, and more users have easier access to downloadable software. Since configuration management is complex already for developers [11], deployment tasks such as installation of valid component configurations will provide a serious challenge to users.

Thus, support for software development processes should emphasize more the communication among heterogeneous widely distributed developers than within individual processes, by providing a common understanding of processes and allowing different processes to be linked together to reflect component dependencies. Similarly, due to increasingly complex dependencies in "systems of systems", better support for software deployment is desirable, particularly since many dependencies can be formalized, allowing some deployment tasks to be automated. Developers should specify and publish component dependencies electronically, since they have more information about their components than anyone else.

Therefore, product structure information such as dependencies should be incorporated already in the software development process, and this information should be made available and used in subsequent deployment of the software product. We thus propose a modeling language for software development processes that uses a common model for both process structure and product structure, and we have designed and implemented a distributed framework which supports both software development and software deployment based on a common interoperable understanding of the software process.

2 OBJECTIVES

Process-centered software engineering environments must support definition and enactment of software development processes, allow different tools to be used for different process steps, and enable multiple participants to work on a process [6]. Distributed software engineering [12] (in the sense of collaborative engineering, not engineering for distributed systems) allows multiple, even widely distributed, developers to participate in individual processes, but typically provides little support for connecting different processes or for working in heterogeneous environments.

We believe distributed software engineering should move one step further and extend the locality scope of processes beyond individual processes, by additionally supporting linking of different processes, including evolving processes and completed ones, within an enterprise, or within a virtual organization, or processes of otherwise independent developers. As a vision, this will form a large "global software process". Process linking reflects dependencies of components under development among each other as well as on existing components, which may be supplied by different developers. Such dependencies precisely and uniquely identify versions and origins of components, including object code, source code, documentation, etc., which is important information for development, e.g. specification, documentation, implementation, compilation, linking, testing, configuration and release, as well as for deployment, e.g. retrieval and installation by users.

Furthermore, we believe that not only the locality scope but also the duration scope of processes should be extended, beyond development time, by using process information in deploying software. Deployment has been identified to encompass configuration, release, installation, and removal of software, both for initial deployment and updates, and involves developers to make software available and users to retrieve software and manage individual user-site software registries [9], while software releasing requires developers to provide dependencies, source location, and further metadata for every software component [10]. Since developers best know about their components, we believe much of this information, particularly dependencies, should be inserted in the process already at development time, with additional information, e.g. for marketing (pricing, source location, etc.), added in the process at release time. Our hypothesis is that deployment and runtime dependencies are a subset of dependencies known already at development time. This does not contradict independent component development, since components only reference those they depend on, but not vice versa. The process thus becomes the primary integrated source of information about its software, allowing the process to be used beyond its traditional purposes, for the full life cycle of a component including development and deployment, instead of dispersing such information in several unrelated databases.

Since development of linked processes occurs in widely distributed heterogeneous organizations, and users similarly represent widely distributed heterogeneous organizations, it is essential that heterogeneous platforms and systems are supported, which demands development and deployment systems to be simple as well as interoperable both among developers and among developers and users. Our objectives are thus to propose a suitable process modeling language and implement it in a distributed system, meeting the three main requirements for processes, i.e. support for heterogeneous organizations and systems, support for process linking, and support for deployment, in addition to traditional requirements, which leads to a simple common interoperable understanding of the process, which we believe is more important than traditional full-featured process representations.

2.1 DIPS Project

Our DIPS project (**D**istributed **I**ntegrated **P**rocess **S**ervices) focuses on the process modeling language and its implementation in the distributed framework, and provides process and tool integration services used by all processes. The term “framework” signifies that in order to perform development activities, the framework must be extended by process-specific tools (editors, compilers, etc.) to obtain a software engineering environment; for deployment activities however, no extension is required. The associated CHIPS project (**C**omponents for **H**ighly **I**ntegrated **P**rocess **S**upport) focuses on high-level semantic data integration by allowing tool components to be specified and generated using compiler-compiler technology. It thus provides process-specific tools which allow component dependency semantics in processes to be formalized, as we believe support for formal methods is increasingly important, particularly when developing reusable components [13]. However, formalizing dependency semantics and using CHIPS tools is optional in DIPS processes, and external tools can be integrated using simple adapters, although on a lower integration level. CHIPS and DIPS together form the GIPSY project (**G**enerating **I**ntegrated **P**rocess support **Sy**stems) [15][16][7].

3 PROCESS MODELING LANGUAGE

Since the goal of a software process is a software product [3], and since we intend to utilize software product information beyond development time and beyond individual processes, the product structure should be reflected in the process structure. We achieve this through structural unity of process and product: the structure of the software product as planned and under development defines the structure of its development process. To this end, our proposed language for modeling processes defines process nodes and dependencies as basic elements for composing processes, it contains definition rules specifying how these elements may be used to compose (define) processes, it contains enactment rules specifying how processes, once defined, may be executed (enacted), it supports product and process evolution, and it allows configurations to be specified on completed process parts for release to other developers and users.

3.1 Software Product

A (software) product is a partially ordered set of artifacts, holding all data which are produced during development and maintenance of the product (contract, specification, implementation, code, test case, documentation, manual, etc.), not just those artifacts delivered to a customer. Every artifact carries the information of a part of the software product and typically depends on other artifacts, thus defining the ordering of the set. Artifacts can exist in different versions (both revisions and variants), and can be part of an enclosing compound artifact, thus defining a tree-like hierarchy with atomic artifacts as leaves. A software product can be understood as a compound artifact, similar to compound documents [17], but extended by dependencies among the versioned parts. An atomic artifact is ideally an object which is an instance of a CHIPS or other object-based tool, that allows it to have formal or informal properties and be related to other artifacts (e.g. an object code artifact depends on its source code artifact), although object adapters also allow conventional files created by external tools to be used as artifacts.

3.2 Software Process

A (software) process is the dynamic view of a software product under development and defines how its artifacts are created, edited and confirmed. A process is a directed acyclic graph (DAG): its nodes are process nodes that represent process steps (the dynamic view of the artifacts) and are either atomic or compound: an atomic process node holds one associated atomic artifact with the tools used and permits only one developer to update it at a time, thus defining the granularity of processes, while a compound process node holds a set of process nodes (atomic or compound), defining a process hierarchy, whose (recursively) associated artifacts make up the compound artifact. The DAG's edges are arrows representing the artifacts' dependencies and point in the direction of development workflow (which is the reverse direction of the dependency).

The process is the hierarchically all-enclosing top compound process node containing all other process nodes with dependencies, and with all versions of the process nodes, and with artifacts attached to their respective atomic process nodes; it represents the all-enclosing artifact making up the whole product. The process structure is the process without attached artifacts. A process represents one software product (typically a software component), and different processes may be linked to represent product dependencies. Developers may thus decide whether a product should be represented as one large or several smaller linked processes, based on their desired degree of interaction.

3.3 Process Dimensions

Processes are defined by creating process nodes and linking them with dependencies to represent the planned product structure, forming a 2-dimensional DAG, and observing process-specific composition rules where imposed by the tools used (e.g. dependency of object code on source code and library artifacts). In order to support variants, dependencies can be grouped in sets of alternatives so that during enactment, only one dependency in the set is used at a time.

Since change is inherent in software processes [18], the language provides a third dimension, the history dimension. When either the process structure or previously created artifacts have to be revised (in process evolution respectively process iteration operations), a new copy of the affected part of the 2-dimensional DAG structure is created (with all nodes and dependencies) and inserted in its place in the top or current layer, while the old copy moves down the history dimension to create a new layer in the process history (fig. 1). Thus, all change is recorded and old revisions of process parts continue to be accessible, though not modifiable (unless copied, although new dependencies may be appended). Revisions may also be used as variants for dependent artifacts, e.g. an artifact may depend alternatively (not simultaneously) on different revisions of an operating system component.

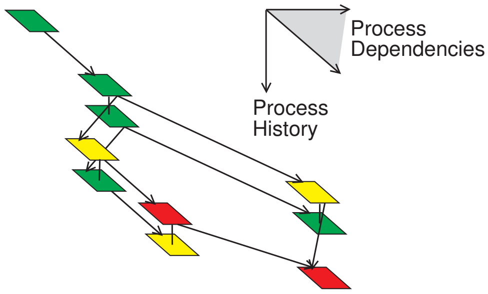

Fig. 1: Process dimensions (graph with nodes and dependencies; artifacts not shown)

3.4 Process Enactment

Defined processes may be enacted immediately by interpretation, there is no separate instantiation or compilation, and processes may continue to be defined even after enactment. During enactment, an atomic process node traverses a sequence of three different states (represented by colors), commencing in state planned:

- planned: no artifact is assigned to the process node;
- editable: an artifact has been created and assigned to the process node and is being developed;
- confirmed: the assigned artifact has a required property specified in the process node, the artifact is confirmed to be completed, and it may not be modified any more.

Confirmation involves manual electronic signing by the responsible developer signifying that the artifact has been checked to fulfill a specified formal or informal property. Where a dependent artifact requires predecessor artifacts for its creation (e.g. creation of object code requires source code to be compiled), this can only proceed after their confirmation. This defines a partial ordering of workflow in the process, visualized by progressing coloring (representing states) of process nodes. State transitions of compound process nodes are similar, but governed their contained process nodes' states.

3.5 Process Linking

In order to link two processes to reflect component dependencies, e.g. a new editor component that depends on a released component framework, an interprocess dependency is established from a node in one process either to a node or to a configuration in another process. Unlike standard dependencies, interprocess dependencies may only be established on confirmed (read-only) nodes and are normally not registered in the nodes they depend on (since these may be in another organization's process where the developer has no write privileges), and therefore definition or enactment operations are not propagated along these dependencies. This is appropriate, since e.g. the editor's developer does not desire his components to be automatically moved to the process history when the framework's developer releases a new revision, but instead continues to use the same one, although he may be notified of the new revision and later decide to make use of it.

An interprocess dependency may also be used as a metaprocess dependency describing an artifact's tool's configuration in the tool's development process, thereby contributing to the precise specification of the software engineering environment's configuration at any time during process evolution.

3.6 Configurations

A configuration is a consistent set of confirmed (therefore read-only) process nodes (with associated artifacts) in a process or in linked processes, used e.g. to define which artifacts belong to a release of a product, specified by a vertical filter projection on the process to select only artifacts of concern (e.g. include object code and documentation artifacts, but no source code), a horizontal projection that defines which revisions of the filtered artifacts are used, and if appropriate a specification of the chosen variants. Consistency rules require that at most one revision (and variant) of an artifact contributes to a configuration. Independently stored configuration sets can be used to describe products consisting of otherwise unrelated configurations.

4 LAYERED FRAMEWORK ARCHITECTURE

In order to provide independence among development and deployment, yet allow processes to be shared, we propose a framework implementing the language based on a layered architecture, with vertical dependencies among layers. As illustrated in fig. 2, basic distribution services are required, upon which common distributed process services are based. These define process services shared among development and deployment and provide process interoperability, while development and deployment systems each extend them with specific features, but both work with processes. Tool integration services allow development tools to be plugged into processes via the common process services.

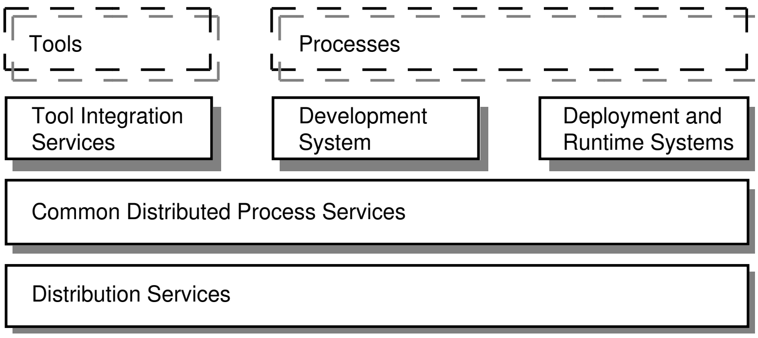

Fig. 2: Four-layered framework architecture

4.1 Distribution Services

Distribution services handle distribution and replication of processes and artifacts, manage access privileges and security, provide a messaging system allowing distributed entities to communicate, provide transactions with a limited form of ACID protection, handle synchronization of replicas, and provide a level of abstraction from location and heterogeneous systems (they should provide the same interfaces and protocols on different platforms). Such services are typically provided by object request brokers for distributed objects extended by specific services, although scalability for Internet-wide distribution demands limits on messaging, transaction and replication protocols.

4.2 Common Distributed Process Services

The common distributed process services handle those aspects of the process modeling language that are common to both development and deployment. They define process nodes (atomic, supporting an associated artifact, and compound) and dependencies (standard and interprocess) as basic process composition elements. Only structural dependency information is handled (predecessor/successor nodes, contained/enclosing nodes, older/newer revision, alternatives), not the optional process-specific dependency semantics handled by tools. Association of artifacts and various attributes (access privileges, time, etc.) with process nodes is handled as well as message forwarding from artifact to artifact. Specification of configurations and configuration sets is supported. Serialization of multi-dimensional process structures for persistent storage and transmission via network is provided. Processes are managed in on-site process bases at developer and user sites (with different access privileges for developers and users), and distribution and replication of nodes and artifacts of processes is supported, but without defining policies. Based on the distributed messaging system, simple messaging protocols are supported that allow distributed developers and users to access and copy processes to carry out development or deployment activities.

Common process services are the key to being able to use a common process representation beyond individual processes and beyond development time, providing a base for process interoperability among developers as well as between development and deployment.

4.3 Development System

The development system implements all the rules for process definition and enactment, allowing processes to be composed, linked, evolved and enacted. It thus defines how the basic elements provided by the common process services may be used to assemble valid processes, how processes evolve when iteration or evolution operations are performed (e.g. copying of affected process parts, establishment of older/newer dependencies, etc.), and enforces constraints on enactment concerning states of process nodes (e.g. no compilation before associated source code is confirmed). It includes a distributed process engine that provides 2D and 3D graphical process views for definition and enactment operations as well as for navigation and query operations on the process structure and artifacts, and based on the simple messaging protocols it implements messaging protocols to support distributed operation. As detailed below, it implements specific distribution policies for process nodes and artifacts. It also provides the user interface for specification of configurations and configuration sets and release of configurations through setting of appropriate access privileges, with every artifact and configuration obtaining a globally unique address that allows precise versioned identification of any copy.

4.4 Deployment and Runtime Systems

The deployment system handles copying of completed and released software products from developers or intermediate sources to users, in the form of configurations and configuration sets. It provides a similar graphical user interface for processes, although with more limited information (e.g. no proprietary information from developers), and all retrieve accesses to processes are read-only. It allows a selected configuration spread over multiple distributed sources to be downloaded in one operation. All retrieved configurations are inserted in a local user-site process base and links established locally reflecting exactly those in the developers' original process copies. This allows validation of configurations and configuration sets at user sites, prevention of invalid installations, and straightforward removal of undesired configurations. Users thus have their own unique base of configurations typically originating from many different processes from different developers.

The runtime system is the host operating system's conventional runtime system with a slight modification: its loader retrieves object code artifacts directly from the local user-site process base, instead of loading files from the local filing system. As with conventional files, access is by name only, i.e. the programming language used for the artifact's implementation does not require any versioning concept, but unlike files, where versions are at best specified by (error-prone) paths, installed valid configurations guarantee that correct versions of all required artifacts are accessed. Ultimately, all local software will be stored in the local process base, and the local filing system will only be used to store data files, or will possibly become obsolete if objects are used for these purposes too. Process bases may also be shared among several user workstations, e.g. diskless network computers.

4.5 Tool Integration Services

Tool integration services allow development tools (editors, compilers, etc.) to be plugged into the process framework in order to extend the framework to a full software engineering environment. The services also define extensible messages for communication among artifacts that are instances of tools. For CHIPS-based tools, specific messages are used that access related artifacts in their process context, to validate formal dependencies or to create new artifacts, e.g. an object code artifact's creation tool may be implemented as a compile message sent to its related source code artifact, which compiles itself and returns the object code artifact. Message types include requests to provide specific views of an artifact, e.g. to create HTML from text-based artifacts used for their WWW presentation. Other tools are supported via adapters, but with limited integration, since our focus is on process interoperability rather than tool integration.

5 DISTRIBUTED FRAMEWORK ARCHITECTURE

This section describes how different distribution policies are used for development and for deployment, for different parts of the framework introduced in the previous section, and for process nodes and for artifacts.

5.1 Development Distribution

In order to actively participate in a process, a developer requires the distribution services, common distributed process services, and development system, and tool integration services if CHIPS-based tools are used, plus the process-specific tools he uses for those artifacts in a process he works on. Furthermore, he typically needs the deployment and runtime systems for process-based software too, unless no such software is used, e.g. in an initial bootstrapping phase.

5.1.1 Development Activities and Process Distribution Policies

A developer typically works together with other developers on a process or on several processes, whereby every developer works on a different part of a process, but also requires information about other parts of a process and about other processes. An analysis of typical operation patterns suggests that non-replicated distribution of artifacts and replicated distribution of all process nodes of a process is a favorable distribution policy, as discussed in [19] (fig. 3). Since developers typically require frequent dependency and other metainformation about related artifacts which is stored in process nodes, but process nodes are updated relatively infrequently, their replication overhead is justified. Artifacts however are distributed according to their developers and are not replicated, since a developer updates his own artifacts relatively frequently (compared to their associated nodes), so this should occur locally, but he reads others less often, and moreover artifacts can be based on very heterogeneous tools which makes replication on heterogeneous hosts difficult. Process node replication is synchronized by specifying a master machine for every process and using a simple primary-backup scheme [14].

Since the number of developers working on the same process is typically small, e.g. 1 - 12 participants (larger projects would typically be run as separate (i.e. non-propagating) linked processes) and processes typically contain less than 500 nodes [19], the number and sizes of messages are relatively small. Since response times are not critical, small-scale replication of process nodes on every developer's machine is feasible, and with reduced response time requirements, even for Internet-wide distribution of developers, e.g. in a virtual organization. More flexible distribution and replication schemes, e.g. with partial or dynamically changing replication, are conceivable, but more complex.

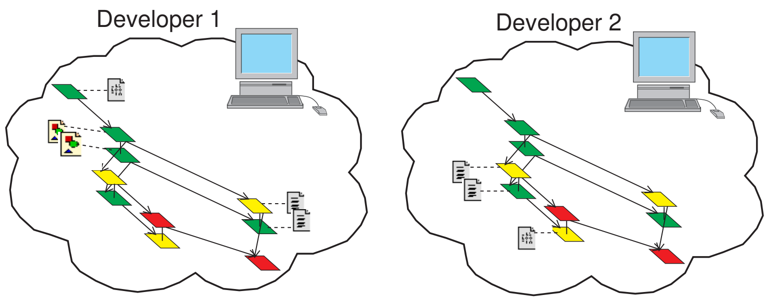

Fig. 3: Distributed development system, illustrating replication of process structure (process nodes) and non-replicated distribution of associated artifacts (shown as icons)

5.1.2 Release Activities

Supported by the development framework and carried out by developers, release comprises specifying a configuration that defines which process nodes have deliverable artifacts, e.g. only object code and documentation but no source code, then setting the access privileges of these nodes with their artifacts and selected attributes to readable for users, and allowing users to connect to the process for retrieval. Released configurations should in principle be made available "forever", although limits are conceivable. For marketing, we also envisage software component releases to be registered in searchable component catalogs on the WWW that carry information such as component purpose and pricing, and include a link to the process, which provides graphical visualization of origin, dependency and configuration information on the WWW, and allows retrieval of selected configurations.

5.2 Deployment Distribution

A user desiring software developed in a process requires the distribution services, the common distributed process services and the deployment and runtime systems, but not the development-related parts of the framework.

5.2.1 Deployment Activities and Distribution Policies

Deployment is carried out by individual users and involves retrieval of released configurations from suitable sources and insertion in the user's local process base. After finding a configuration, e.g. in a WWW-based software catalog, its globally unique address serves to precisely identify it at whichever source it is made available, and it also identifies the original source that made it available, the developer. Typically, a developer will initially deploy copies on his enterprise's process server via its intranet, from where other companies will deploy copies on their own process servers via Internet, which again will be accessed via intranets for their employees' local deployment (fig. 4). Thus, in order to avoid excessive load on a developer's site, this requires a redirection mechanism for deployment systems, where every deployment system has its list of preferred servers to be consulted for retrieval of configurations, possibly transitively, before the developer's original site has to be contacted if unsuccessful elsewhere. If copyright or licensing restrictions are to be enforced, then most individual users would typically not have access to the original developer's site but only to their enterprise's copies, thus requiring redirection.

Users thus build their own process base of configurations, and any such process base may again become a server for other users, such as the enterprise servers, and a developer may even be a user of his own releases (since all releases are read-only), in which case no copying is required. For the management of servers and process bases, different schemes are envisageable, from cache management to manual management. Since deployment involves only read-only copies that are not updated, wide-scale replicated deployment scales well.

Although not a requirement, an advantageous complement to our deployment scheme would be an endogenous ownership mechanism for software as proposed in [1] which would allow e.g. a pay-per-use scheme for software instead of the conventional pay-per-copy scheme. This would allow user access privileges to be greatly simplified as all software could be copied freely and instead only its execution or instantiation would be charged.

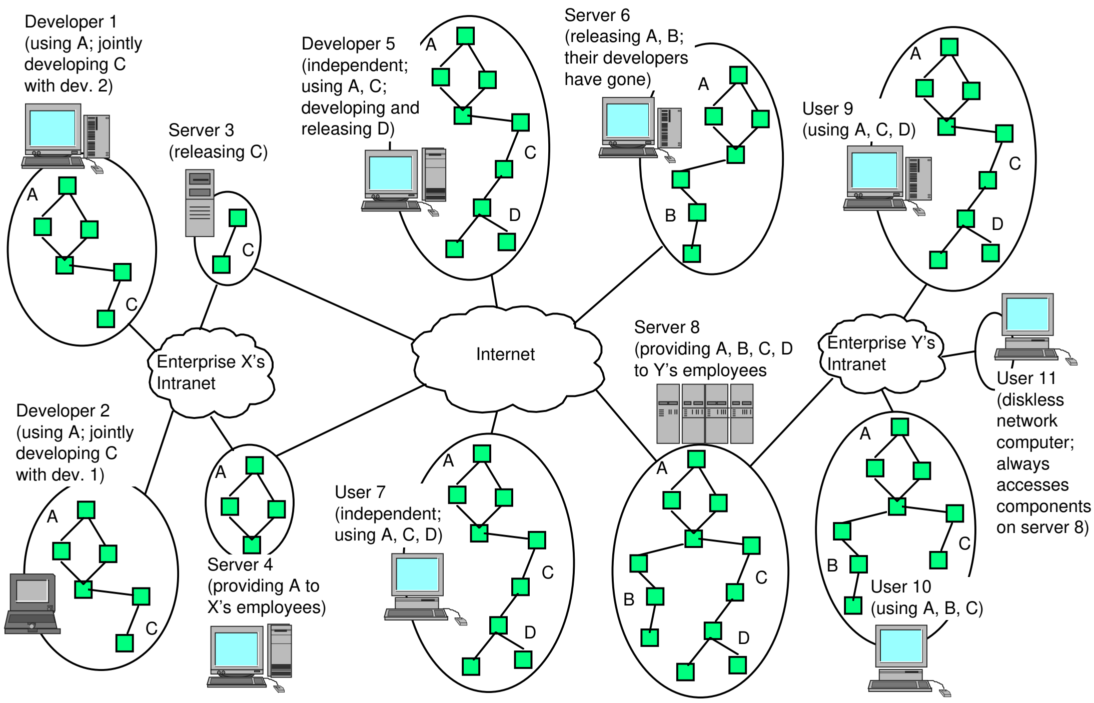

Fig. 4: Internet and intranet development and deployment scenario (involving 4 components: A, B, C, D with simplified linked processes; versions and artifacts not shown)

6 IMPLEMENTATION

We have implemented a limited-functionality prototype system of the distributed framework in Oberon System 3 [8], running on both Power Macintosh and UNIX (Sun SPARCstation) systems, together with some CHIPS-based tools (text editors, compilers) and adapters for external tools (text and graphical editors, wordprocessors; currently limited to tool invocation). Distributed operations are based on TCP/IP to potentially allow Internet-wide distribution. Oberon was chosen several years ago, but meanwhile CORBA systems and Java have emerged which fulfill many of our basic requirements and may be used in the prototype's next major revision; we also believe the framework should eventually be integrated in the Internet to facilitate Internet-wide collaborative work [4]. We have defined simple messaging protocols (e.g. for process replica synchronization, messaging among distributed objects such as artifacts, deployment requests by users, etc.) which run on specific TCP ports. Artifacts are currently addressed by their host's Internet address, process name, and process node number, all of which are unique within their scope; configurations are addressed by their unique name within a process; more elaborate addressing schemes are envisaged. As a demonstration of providing process information on the WWW to help users making retrieval choices, we have also built a dedicated WWW server that visualizes current process dependencies on-the-fly in 2D and 3D graphics on the WWW (i.e. independently from the framework), providing GIF and Apple QuickDraw 3D graphics, and showing text-based artifacts in HTML containing hyperlinks to the correct versions of depending artifacts [20][7].

The prototype allows distributed operation for graphical process definition and enactment including process iteration and evolution providing artifact revisions, and specification and validation of configurations, but variants, configuration sets and process linking are not yet supported. Meanwhile, some process linking aspects are simulated e.g. by defining two component processes in one large process. We have used the system successfully to support software processes in student exercises of lectures at ETH (as in fig. 5), and we have also partially applied it to its own development in a bootstrapping manner (involving up to 4 developers, and 60 process nodes in a process with up to 8 revisions each). Whenever we complete a new release, an appropriate configuration is defined, related access privileges are set to readable for all (release access privileges are not yet differentiated), and a process copy with all required artifacts can be downloaded automatically by another user from all involved distributed sources at his request (currently only from the original sources, without redirection), while development of new revisions in the same process continues unaffected.

In order to use the development and/or deployment system, no modifications are required to an operating system it runs on. For the prototype, no modifications are required to the local runtime system either, since deployed artifacts are stored as files, and components can thus be instantiated conventionally, assuming modern dynamic linking and loading capabilities as in the Oberon system [8]. However, we envisage user-site process bases to be handled by object management systems separate from filing systems, as processes and artifacts remain objects even after deployment and thus retain their process-awareness. Artifacts will only be accessible via process instead of inconsistently via both process and filing system, i.e. process and artifact concepts will replace directory and file concepts for software, handling much more metainformation about components (e.g. dependencies) than filing system directories, and requiring the loader to instantiate objects directly out of the process base instead of from file.

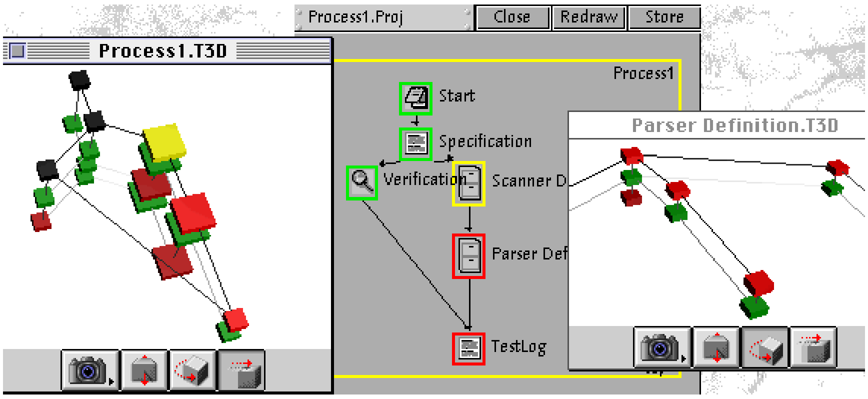

Fig. 5: Screenshot of a process on the prototype in 2D (middle) and 3D views (left: top-level process hierarchy; right: zoom-in on a compound node)

7 DISCUSSION

7.1 Related Work

Process-centered software engineering environments have been studied extensively in recent years, with papers collected e.g. in [6] and [5]. There is a great diversity in process modeling languages and paradigms, with many systems providing complex languages to manage individual processes, but little support for connecting different processes or for managing processes in heterogeneous widely distributed systems such as in virtual organizations, where a simple interoperable process base is more important than sophisticated features. Support for software deployment is also lacking, since this is typically regarded as a separate task.

As a survey of release management, installation and configuration registry systems discusses, only one research system covers all deployment activities [9], and release management systems typically lack either adequate support for dependency management or distribution [10], and unlike in our system, dependencies are typically specified manually. While many systems support individual development or deployment activities, we are not aware of any integrated system that directly employs process information for dependency and configuration management, and which supports software development in widely distributed heterogeneous organizations, configuration management, release and installation via network, and management of user-site process bases (registries) with instantiation directly from there instead of via filing system, as in our system.

7.2 Conclusions and Outlook

Regarding the new challenges to software development and deployment provided by wide-scale Internet connectivity, emerging distributed component technologies, and virtual organizations, we believe more emphasis should be placed on organizational interoperability in wide-scale systems and in particular in process interoperability among developers as well as among developers and users, since they concern much of the same information and would benefit greatly from being able to share it in a straightforward manner. As there is apparently no existing system focusing on such objectives, we do cannot investigate interoperability with existing systems but instead propose a new approach to providing interoperability among developers and among developers and users, applicable to wide-scale distributed systems which are inevitably heterogeneous.

The key idea is to use the software development process to hold product structure information such as component dependencies that concerns developers of different processes as well as users installing and configuring “systems of systems”, and we propose a process modeling language that manages product and process structure, state and evolution. Based on this, we present a distributed framework with a process support system that allows integration of both development and deployment systems, enabling different developers and developers and users to interoperate by sharing common process representations. This is only possible by providing an interoperable distributed system for process representation and management. Ultimately, we believe that this will lead to the “global process” of all linked processes, replacement of filing system by object management concepts to store software, improved Internet integration of process support, and also provide opportunities for agent-based software deployment. We present a successful prototype as proof-of-concept, and we believe that based on these ideas, definition of standards and systems should be pursued leading to global process interoperability.

REFERENCES

[1] B. Cox, Objects as Property, *IEEE Software*, 14(1):22-24, Jan. 1997.

[2] W. Davidow and M. Malone, *The Virtual Organization: Structuring and Revitalizing the Corporation For the 21st Century*, Burlingame Books, 1992.

[3] P.H. Feiler and W.S. Humphrey, Software Process Development and Enactment: Concepts and Definitions, *Proc. 2nd Int’l Conf. Software Process*, IEEE CS Press, 1993, 28-40.

[4] R.T. Fielding and G. Kaiser, Collaborative Work: The Apache HTTP Server Project, *IEEE Internet Computing*, 1(4):88-90, July/Aug. 1997.

[5] A. Finkelstein, J. Kramer, and B. Nuseibeh (eds.), *Software Process Modelling and Technology*, Research Studies Press/Wiley, 1994.

[6] P.K. Garg and M. Jazayeri, *Process-Centered Software Engineering Environments*, IEEE CS Press, 1995.

[7] GIPSY project home page (with CHIPS and DIPS), <http://www.tik.ee.ethz.ch/~gipsy/>, 1997.

[8] J. Gutknecht, Oberon System 3: A Vision of a Future Software Technology, *Software–Concepts and Tools*, Springer, 15(1):26-33, 1994;
also: Oberon System 3 home page, <http://www.oberon.ethz.ch/system3/>, 1997.

[9] R.S. Hall, D. Heimbigner, A.v.d. Hoek, and A.L. Wolf, An Architecture for Post-Development Configuration Management in a Wide-Area Network, *Proc. 17th Int’l Conf. Distributed Computing Systems (ICDCS)*, Baltimore, IEEE CS Press, May 1997, 269-278.

[10] A.v.d. Hoek, R.S. Hall, D. Heimbigner, and A.L. Wolf, Software Release Management, *Proc. 6th European Software Engineering Conf. (ESEC)/5th ACM SIGSOFT Symp. Foundations of Software Engineering (FSE)*, Zurich, Switzerland, Springer LNCS 1301, Sept. 1997, 159-175.

[11] D.B. Leblang, The CM Challenge: Configuration Management that Works, in: W.F. Tichy (ed.), *Configuration Management*, Wiley, 1995, chapter 1, 1-37.

[12] C.W. Loftus et al., *Distributed Software Engineering*, Prentice Hall, 1995.

[13] B. Meyer, The Next Software Breakthrough, *Computer*, 30(7):113-114, July 1997.

[14] S. Mullender (ed.), *Distributed Systems*, 2nd ed., ACM Press/Addison Wesley, 1993.

[15] T. Murer and D. Scherer, Structural Unity of Product, Process and Organization Form in the GIPSY Process Support Framework, *Proc. 8th Conf. Software Engineering Environments (SEE)*, Cottbus, Germany, IEEE CS Press, Apr. 1997, 93-100.

[16] T. Murer and D. Scherer, Organizational Integrity: Facing the Challenge of the Global Software Process, *TIK-Report No. 51*, TIK Laboratory, ETH Zurich, July 1998.

[17] R. Orfali, D. Harkey, and J. Edwards, *The Essential Distributed Objects Survival Guide*, Wiley, 1996.

[18] D.L. Parnas and P.C. Clements, A Rational Design Process: How and Why to Fake It, *IEEE Trans. Software Eng.*, 12(2):251-257, Feb. 1986.

[19] D. Scherer, T. Murer, and A. Würtz, Designing the Distributed Architecture DIPS for Cooperative Software Engineering, *Proc. 30th Hawaii Int'l Conf. System Sciences (HICSS)*, IEEE CS Press, Jan. 1997, Vol. I, 150-158.

[20] D. Scherer, T. Murer, and A. Würtz, Towards Providing Software Component Interoperability Information on the WWW, *Proc. 2nd Australian WWW Conf. (AusWeb)*, Gold Coast, Australia, July 1996, 109-116; also at: <http://www.scu.edu.au/ausweb96/tech/scherer/paper.html>.

TIK-Reports

A Relational Data Base Design for an X.500 Directory System Agent

F. Perruchoud, C. Lanz, B. Plattner, July 1990, TIK-Report No. 1

Model and Functionality Definition for the Collaborative Editing Conferencing System MultimETH.

H. Lubich, July 1990, TIK-Report No. 2

X.400 Security Capabilities: Evaluation and Constructive Criticism

M. Müller, August 1990, TIK-Report No. 3

CPU Evaluation for ADaM

Schibli, M. Tadjan, Januar 1992, TIK-Report No. 4

Aspekte computergestützter Kooperation - Schriftliches Material eines Seminars an der ETH Zürich

Hannes Lubich, Januar 1993, TIK-Report No. 5

Extensible Attribute Grammars

R. Marti, T. Murer, December 1992, TIK-Report No. 6

GIPSY: A Generator for Incremental Programming Systems

R. Marti, T. Murer, June 1992, TIK-Report No. 7

Test Case Validation - TTCN Test Case Validation Against SDL Specifications

F. Kristoffersen, T. Walter, May 1994, TIK-Report No. 8

Conformance and Interoperability - A critical assessment

T. Walter, B. Plattner, September 1994, TIK-Report No. 9

OOP-Softwarearchitektur für Multimediakommunikation

Serge Hoffmann, 1995, TIK-Report No. 10

A Comparison of Selection Schemes used in Genetic Algorithms

Tobias Blickle, Lothar Thiele, April 1995, TIK-Report No. 11

Spezifizieren und Generieren von integrierten Umgebungen mit GIPSY

A. Helbling, T.Murer, Mai 1995, TIK-Report No. 12

Die Spezifizierungssprache GIPSY/L

A. Helbling, T.Murer, Mai 1995, TIK-Report No. 13

SCSM - Synchronous Composition of Sequential Machines

H. Fierz, Juni 1994, TIK-Report No. 14

Specification of GMS Access Protocol (GAP) Version 1.0

Erik Wilde, March 1996, TIK-Report No. 15

System-Level Synthesis Using Evolutionary Algorithms

Tobias Blickle, Jürgen Teich, Lothar Thiele, April 1996, TIK-Report No. 16

A Flow-Based Approach to Solving Resource-Constrained Scheduling Problems

Jürgen Teich, Lothar Thiele, July 1996, TIK-Report No. 17

Multi-User Multimedia Editing with the MultimETH System

Erik Wilde, February 1994, TIK-Report No. 18

Specification of GMS System Protocol (GSP) Version 1.0

Erik Wilde, September 1996, TIK-Report No. 19

A Proposal for a Real-Time Extension of TTCN

T. Walter, J. Grabowski, 1996, TIK-Report No. 20

OberonT -- eine Programmiersprache für sicherheitskritische Systeme

D. Schweizer, 1996, TIK-Report No. 21

GIPSY: Generating Integrated Process support SYstems - Project Overview

T. Murer, A. Würtz, D. Scherer, D. Schweizer, December 1996, TIK-Report No. 22

CHIPS Reference Manual

A. Würtz, T. Murer, October 1996, TIK-Report No. 23

Dynamic Min-Max Problems

L. Thiele, January 1997, TIK-Report No. 24

Da CaPo++ - Communication Support for Distributed Applications

B. Stiller, D. Bauer, G. Caronni, C. Class, C. Conrad, B. Plattner, M. Vogt, M. Waldvogel, January 1997, TIK-Report No. 25

NFS Performance Investigations Based on Classical IP over ATM

T.V. Prabhakar, D. Bauer, B. Stiller, June 1997, TIK-Report No. 26

Dynamic Semantics of the Oberon Programming Language

P.W. Kutter, December 1996, TIK-Report No. 27

Project Da CaPo++, Volume I: Architectural and Detailed Design

B. Stiller, D. Bauer, G. Caronni, C. Class, C. Conrad, B. Plattner, M. Vogt, and M. Waldvogel, July 1997, TIK-Report No. 28

Project Da CaPo++, Volume II: Implementation Documentation

B. Stiller, D. Bauer, G. Caronni, C. Class, C. Conrad, B. Plattner, and M. Waldvogel, August 1997, TIK-Report No. 29

Traffic Shaping in an ATM Environment

Prabkahar T V, Burkhard Stiller, Thomas Walter, 1997, TIK-Report No. 30

Synchronisation Issues in Distributed Applications: Definitions, Problems, and Quality of Synchronization

Christina Class, 1997, TIK-Report No.31

Optimized Software Synthesis for Digital Signal Processing Algorithms - An Evolutionary Approach

Juergen Teich, Eckart Zitzler, Shuvra S. Bhattacharyya, 1997, TIK-Report No.32

SCF - State Machine Controlled Flow Diagrams

Lothar Thiele, Jürgen Teich, Martin Naedele, Karsten Strehl, and Dirk Ziegenbein, January 1998, TIK-Report No.33

A set macro language for ASMs

Philipp Kutter, January 1998, TIK-Report No.34

SCF - State Machine Controlled Flow Diagrams

M. Anlauff, P.W.Kutter, and A. Pierantonio, Januar 1998, TIK-Report No.35

Telepoly++ - Towards a Highly Distributed Synchronous and Interactive Teleteaching Environment

Thomas Walter, Hans Hänni, January 1998, TIK-Report No.36

Analysis of the Concord Algorithm and the Adaptive Synchronization Protocol Using QoSy

Christina Class, January 1998, TIK-Report No.37

HAM: Hardware Moved Molecules, Annual Report 1997

Martin Gerber, Thomas Gössi, February 1998, TIK-Report No.38

Introducing Design Patterns for Petri Nets

Jörn Janneck, Martin Naedele, February 1998, TIK-Report No.39

Symbolic Model Checking of Petri Nets Using Interval Diagram Techniques

Karsten Strehl, Lothar Thiele, February 1998, TIK-Report No.40

Efficient Security for Large and Dynamic Multicast Groups

Germano Caronni, Marcel Waldvogel, Dan Sun, Bernhard Plattner, April 1998, TIK-Report No.41

Project Da CaPo++, Volume III: Performance Evaluation

B. Stiller, G. Caronni, C. Class, C. Conrad, B. Plattner, M. Waldvogel, February 1998, TIK-Report No. 42

An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach

Eckart Zitzler, Lothar Thiele, April 1998, TIK-Report No.43

The WaveVideo System and Network Architecture: Design and Implementation

George Fankhauser, TIK-Report No.44

Integrating Domain Specific Language Design in the Software Live Cycle

P.W. Kutter, D. Schweizer, L. Thiele, May 1998, TIK-Report No.46

Object-based Abstract State Machines

P.W. Kutter, J. Janneck, May 1998, TIK-Report No.47

Swiss German Polyphone -- Schlussbericht

K. Huber, June 1998, TIK-Report No.48

Mapping Automata - Simple Abstract State Machines

Joern W. Janneck, Philipp W. Kutter, TIK-Report No.49

Object-based Mapping Automata - Reference Manual

Joern W. Janneck, TIK-Report No.50

Organizational Integrity: Facing the Challenge of the Global Software Process

T. Murer, D. Scherer, TIK-Report No.51