Combining Control and Data Integration in the SPADE-1 Process-centered Software Engineering Environment

Sergio Bandinelli Alfonso Fuggetta Luigi Lavazza Gian Pietro Picco
CEFRIEL and Politecnico di Milano*
Via Emanueli 15, 20126 Milano (Italy)

Abstract

For Process-centered Software Engineering Environments (PSEEs) it is essential to provide the set of tools that are needed to actually produce software in practice.
SPADE-1 is a PSEE originally designed in order to allow the developer to use regular Unix tools (such as `vi` *and* `cc`*) and integrated environments based on the selective message broadcast (like DEC FUSE). However, this approach did not allow tools (and users) to share common structured data.*
This paper describes the mechanism that has been introduced in SPADE-1 in order to achieve both control-oriented integration and data-oriented integration of tools. In this way tools can share structured data in an ordered and controlled way.

1 Introduction

SPADE is a research project carried out at CEFRIEL and at Politecnico di Milano with the goal of developing an environment for Software Process Analysis, Design and Enactment. SPADE is centered on a process modeling language called SLANG (SPADE LANGuage), an extension of a high-level Petri nets [2, 3].

SPADE-1 is the first implementation of the SPADE concept[1] The design of SPADE-1 is based on the idea that the paradigm of the process modeling language can be reasonably kept distinct from the paradigm used to guide the process agents and to interact with the users and the development tools. This separation of concerns is useful to support distribution, evolution and improvement of the development environment, and to integrate different kinds of user interaction paradigms in the same PSEE. Therefore, SPADE-1 presents two main components: the *process enactment environment* and the *user interaction environment* (including the development tools), connected by a *filter* that manages the communication between them [1]. In SLANG it is possible to model both activities that are carried out automatically by the process interpreter, and activities that require the intervention of human agents. The behavior of people and tools involved in the production process can be modeled consistently in the process model by means of two constructs:

- The *black transitions* invoke programs that exist outside the process enactment environment (e.g., the Unix editor `vi`).
- The *user places* are places where tokens appear not as a consequence of some transition firing, but because of some event occurring outside the process enactment environment.

In this way, the process model contains the knowledge about available tools, what they do, what parameters and options can be included in their invocation, etc.

In this scenario it is clearly important to achieve integration among tools and with the process environment.

SPADE-1 version 1.1 obtained tool integration using DEC FUSE [6], a message-based, service-level integrated tool environment. In this paper we discuss the limitations of this approach to tool integration when it is embedded into a PSEE. We also present an additional mechanism (built in the current version of SPADE-1) to overcome these limitations, and in particular to achieve data integration among tools and the process environment.

2 Tool integration

In SPADE-1, tools are integrated by means of DEC FUSE. DEC FUSE belongs to the generation of tool environments that was originated by FIELD [8]. In

*Contact information: E-mail:`lavazza@mailer.cefriel.it`, Tel: +39-2-66100083, Fax: +39-2-66100448

[1] SPADE-1 version 2.2 is currently available.

these environments, tools are viewed as collections of services, that can be invoked by multicasting messages. Tools can therefore issue messages to require services, to notify changes in their state, to acknowledge requests, etc. For example, when the compiler issues a message announcing that an error has been found, the editor can react to the message by opening the erroneous source file and positioning the cursor onto the offending line. When the file is saved, the compiler is notified via a message and can retry the compilation.

In SPADE-1 each message issued by the tools is captured by the filter that interfaces the process environment and the user environment. The filter transforms the message into a token that is inserted in a user place. The presence of the new token will eventually cause the process interpreter to react according to what specified in the process model. Tools can receive messages only from the process enactment environment. A message is sent by invoking a simple message-sender program through the black transition construct. In this way, the process environment takes full control over tool execution. This situation brings three relevant benefits:

- The process knowledge can be removed from individual tools, and the policy concerning the use of tools can be made explicit in the process model.
- The process model can specify not only the high-level activities involved in the production process, but it can be detailed (if needed) to specify the elementary actions that tools (and humans who use the tools) have to perform.
- While DEC FUSE is a single-user tool, the process environment can act as a bridge that connects different instances of DEC FUSE. In other words, each user can interact with other users in a controlled and uniform way by means of the process. For example, a user can update the header file of a module by means of the DEC FUSE editor: the process receives the save-file request and — while granting the service — sends the proper messages to the users of the updated module. These messages are in general invocations of DEC FUSE tool services that are meant to maintain the consistence between module definition and module usage.

The scenario described above can be considered appropriate as long as traditional tools (such as `vi`, `cc`, `make`, `RCS`, etc.) are considered. In this case, messages will contain little more than a command (such as "OpenFile") and a parameter (e.g., the file name).

Data-integration Very often it is useful that tools share data structures. For example, a syntax-directed editor and a compiler can share the same abstract syntax representation of the source code. A discussion of data integration in software engineering environments can be found in [9].

OODBMS are particularly suited to support data integration tools, since they can manage complex objects (that can be seen at different granularity levels), persistency, multi-user access control, etc. For this reason, and being SPADE-1 built on the O_2 OODBMS [5], it was a natural choice to use O_2 as a base also to introduce data integration[2].

In SPADE-1, process data are represented as (typed) tokens in a Petri net. Each token is implemented as an O_2 object. These data are passed to tools to pursue specific goals such as compiling a source code or building a configuration.

Process engines and tools are concurrent processes that use a (common) O_2 database, and are thus O_2 clients[3]. In order to achieve data-integration we had to devise how to pass data among clients. Data are actually stored in the common database, thus they are accessible to every client: the real problem is to give a client the visibility of some data which it is not aware of.

- Passing a reference (e.g., via DEC FUSE) is not possible, because this reference (i.e., the object identifier) is not known (it exists in the O_2 server, but it cannot be accessed by an O_2 client) and, moreover, the address spaces of the two clients are different.
- Passing the whole object through the DEC FUSE communication mechanism — that supports only plain character sequences — is also hardly feasible, since it would be necessary to unparse the object into a string, transmit it, then parse the string and finally rebuild the object. This is an inefficient operation, and, moreover, it would produce a *copy* of the object.

The solution we devised is illustrated in next section. It is a refinement of a solution that is rather trivial in a database application: two clients can communicate via a shared object. In O_2, a *named* object is persistent and is visible by all clients that use the schema where the object name is declared. Therefore, it is sufficient to declare a named object in the SPADE

[2] Even though we refer to O_2, the mechanism described in this paper is applicable to most OOBDs.

[3] O_2 is based on a client-server architecture.

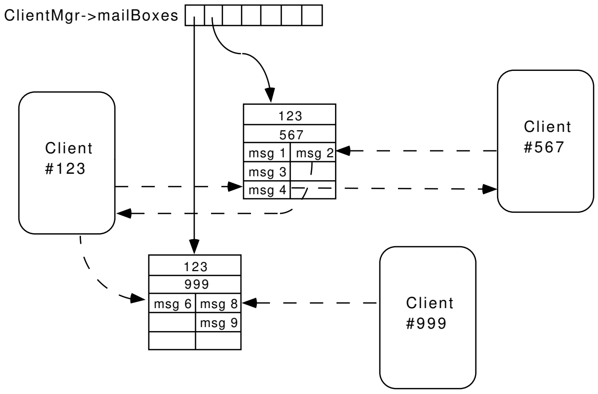

Figure 1: The Inter-Client Communication Mechanism.

schema to let SPADE clients (i.e., the process interpreters and tools) communicate. The problem with this solution is that only one named object cannot support the whole communication traffic (the object has to be locked exclusively when used, thus preventing parallel communication). Using several objects (e.g. one per tool type) does not solve the problem, and moreover forces the communication objects' names to be hard-coded in the schema, thus requiring schema update when adding a tool to the PSEE.

3 Data and control integration in SPADE-1 v.2.0

Figure 1 shows the Inter-Client Communication Mechanism (ICCM), that we designed to improve the communications among clients [4]. `ClientMgr` is a named object in the O_2 clients' schema, and is therefore persistent and visible to all clients. `ClientMgr` manages a list of *mailboxes*, the objects used for actual communication, and provides services to clients requesting a connection through `ClientMgr`'s methods.
The mailboxes serve as the communication channels through which information flows from a client to another. Each mailbox has two message queues, containing the messages exchanged by the communicating clients in both directions.

As already mentioned, `ClientMgr` is a named object in the clients' schema. Mailboxes, being attached to `ClientMgr`, are thus persistent and visible by the clients.
When a client A wants to set up a communication channel with another client B, it creates a mailbox M, and posts a connection message. If B does not exist it is then created. Upon creation B is informed of the identity of its creator and of its own identity: this pair of identifiers is used as a key to search the list of mailboxes for the mailbox that lets B communicate with A. If B already exists, A uses an existing mailbox connected to B in order to inform B that there is a connection request, and that the requests was originated by client A. In both cases, when B has retrieved the connection message from M, the mailbox is dedicated to the communication between A and B. Note that the two clients can keep a local direct reference to mailbox M, thus achieving fast access, i.e., it is never necessary that A and B search again for M in the list of mailboxes. Since every read/write operation on a persistent object has to be performed within a transaction, the situation described above brings an important benefit: when a client sets up a communication with another client[4], it has to lock the whole list of mailboxes. Once the set up is completed, however, subsequent message postings and retrievals can be accomplished by locking in an exclusive mode just M. In other words, we use `ClientMgr->mailBoxes` only to ensure persistency and visibility of mailboxes to all the clients, but we can lock just one mailbox at a time, thus maximizing parallelism [5].

The mechanism sketched above provides great flexibility in building communication schemes. In particular, we have three possibilities: to use DEC FUSE alone, as in SPADE-1 v.1.1, to combine ICCM with DEC FUSE, or even to rely exclusively on ICCM. In the second case FUSE messages could bring commands and notifications, while process data (e.g., the software artifacts) are passed via mailboxes. The integrated tools would retain the properties of FIELD-like tools, but they would no longer rely on the file system to store persistent data. Instead, they would share typed data with other tools.

The availability of ICCM makes it possible to develop PSEEs that achieve tool integration without using FIELD-like environments at all. In this case the process engines and the tools are just processes that communicate via mailboxes (it is only necessary to define the appropriate data type `message`, in order to let mailboxes transmit commands and notifications as well as process data). The advantages of control integration are still retained.
The CORBA [7] approach to object request brokerage could be required when SPADE evolves to support heterogeneous environments. For the time being, i.e., as

[4]This is a relatively unfrequent action in SPADE-1.

[5]This will have an even greater impact when O_2 will support object locks. In fact, locks are currently placed on pages, making possible spurious transaction conflicts. O_2Technology is steering towards object locking.

long as SPADE is bound to O_2, that imposes its architecture, concurrent access mechanisms, language, etc., ICCM is a much simpler (and more efficient) solution for tool communication.

4 Concluding remarks

In this paper we presented the integration of tools in the SPADE-1 process centered software engineering environment. In particular a mechanism that achieves both control integration and data-integration was described. By means of this mechanism, tools can be controlled at the service level via selectively multicasted messages and can share data (i.e., persistent objects stored in the O_2 OODBMS, in the case of SPADE-1).
The process model has a strong control over tools, while tools can share the process data in a disciplined and controlled way.

We believe that the availability of ICCM gives us the means to achieve nearly all the tool integration properties described in [9].

Acknowledgments This work is being carried out within and partially supported by the ESPRIT III project 6115 GOODSTEP (General Object-Oriented Database for the Software Process).

References

[1] S. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza. The Architecture of the SPADE-1 Process-Centered SEE. In *3rd European Workshop on Software Process Technology*, Grenoble (France), February 1994.

[2] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. The SLANG 1.0 Process Modeling Language Reference Manual. Technical Report RT93032, CEFRIEL, Via Emanueli, 15 - 20126 Milano (Italy), September 1993.

[3] Sergio Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi. Software Process Model Evolution in the SPADE Environment. *IEEE Transactions on Software Engineering. Special Issue on Process Evolution*, December 1993.

[4] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing and Implementing Inter-Client Communication in the O_2 Object-Oriented Database Management System. In *Proceedings of the International Symposium Object-Oriented Systems, Methodologies and Applications*, Palermo (Italy), September 1994. AICA.

[5] O. Deux. The O_2 System. *Communications of the ACM*, 34(10), October 1991.

[6] Digital Equipment Corporation, Maynard, Massachusetts. *DEC FUSE Handbook*, December 1991. Version 1.1.

[7] OMG. The Common Object Request Broker: Architecture and Specification. Document 91.12.1, OMG, 1991. Rev. 1.1.

[8] S. Reiss. Connecting Tools using Message Passing in the FIELD Program Development Environment. *IEEE Software*, pages 57–67, July 1990.

[9] Ian Thomas and Brian A. Nejmeh. Definition of tool integration for environments. *IEEE Software*, 9(2):29–35, March 1992.