Do Process-Centred Environments Deserve Process-Centred Tools?

Wolfgang Emmerich and Anthony Finkelstein

Dept. of Computer Science, The City University,
Northampton Square, London EC1V 0HB, UK
{emmerich|acwf}@cs.city.ac.uk

Abstract. Process-centred software development environments integrate a process engine, which enacts a process program, with tools that automate particular tasks or provide facilities for document production. Previous papers in this workshop series have focussed on process automation from a process programming point of view and have discussed language primitives and techniques for their enactment. They assumed implicitly that off-the-shelf tools were to be integrated into these environments. The degree of support that can be achieved, however, is limited by the integration facilities offered by these tools. We consider the problem from a different point of view and investigate the implications of fine-grain process support for tool construction.

1 Process-Centred Software Development Environments

Process programming languages provide primitives for expressing communication and coordination of tasks that are to be performed by multiple developers cooperating in a software process [3]. Process programming languages are used to formalise processes into *process programs*. Process-centred software development environments (PSDEs) have one or multiple *process engines* that enact these process programs. Process engines have either been compiled from a process program and been linked with a run-time environment (e.g. APPL/A [16]) or they interpret process programming languages (e.g. Merlin [11], Marvel [12] and SPADE [2]). In both cases, process engines perform single process automation steps by means of *services*, they request from *tools*, to be integrated into the PSDE. An example of such a service request is the creation of a new document. Furthermore, process engines have to update the state of a process program on the basis of *process-sensitive events*, that can often only be detected by tools. An example of such a process-sensitive event is the change of a document that requires the creation or deletion of some other document.

Some coarse-grained service requests may be implemented by starting a *black box tool*, that performs a particular computation and returns the result to the process engine upon termination. A compiler is a classical example of this. As suggested in [17], enveloping techniques can be used to hide the details of tool invocation from the process program. To improve the degree of process automation and user guidance the process environment provides, however, coarse grained services, as they are provided by enveloped black box tools, are not appropriate.

The process engine has to use more information than just the return code, in particular from interactive tools such as editors and browsers. The engine also has to be able to invoke more specific services than starting a tool. This means that a sequence of fine-grained services need to be performed between start-up and termination of a tool and during its course of execution, a sequence of event occurrences have to be communicated to the process engine.

As an example, consider access rights. As they will have to be changed fairly frequently, access rights should be maintained by process programs. Now assume a KAOS requirement specification [5], which consists of different documents for goals, constraints, actors, entities and relationships. These documents are associated with each other through various relationships, but owned by different requirement engineers. The process program might define a strategy for access rights that grants owners the right to modify their documents and all other engineers working on the project the right to read their contents. A PSDE including tools for all these documents will have to support developers in browsing through the various relationships to understand the relationship of a document to associated documents. While doing so, tools must respect the access rights defined in the process program. This means that browsing has to be implemented jointly by process engine and tools. There are several ways of achieving that. Quite often access rights are checked using facilities of the database system that is used to store documents. Communicating access rights via a database, however, would require the process program to have intimate knowledge about the schema chosen for the persistent representations of all the documents since the access rights to any part of the document have to be changed. This is not desirable. A more favourable option, is to control access with a communication protocol between process engine and tools. In this protocol tools would issue an `open` event to the process engine as soon as they recognise that a developer is browsing through a relationship that leads to another document. The engine would then reason about the developer's current access rights and respond either by requesting an `edit` service from the respective tool if the developer is allowed to modify the document or else by requesting a `view` service, which will open the document but prevent the developer from changing it.

2 Architectures for Fine-grained Process Guidance

Hardly any off-the-shelf tool offers facilities for such a communication protocol, let alone a proper interface to its internal document representation. This observation is particularly true for tools in CASE environments that use proprietary databases, such as Teamwork or Maestro II. More sophisticated environments, such as Software through Pictures [18], offer an application programming interface on the basis of a tool integration framework, such as Hewlett Packard's Softbench [4] or Sun's ToolTalk [15], through which services can be invoked. It is, however, fairly common that the particular services needed by a process program are not offered at the programming interface.

We see four different architectural options for how more fine-grained process

guidance can be achieved. They all require to drop the assumption that off-the-shelf tools are to be integrated. The first option is to break down complex tools into a "grey soup" of very simple tools that perform the services desired and can be either invoked from a common user interface or from the process engine. The second option are *process-centred* tools, which have been constructed or customised with a generation facility so that they support the set of those services and events required in a particular process program. Rather than starting them up and awaiting their termination, the process engine would communicate with these tools via some distributed computing framework. With these first two options, the process engine would remain in complete control of the process. If we also drop this assumption, the third option is *process delegation*, where a process program would down-load a fragment of the overall process into tools. These would then enact the fragment and provide the fine-grained guidance desired. The last option is to decentralise process models completely and execute them within tools rather than in a dedicated process engine.

An assessment of these four options is complicated by the fact that they have been explored at very different levels of detail. The approach of process-centred tools was pursued in the GOODSTEP project and is discussed below. An experiment with completely decentralised process models was conducted in the Viewpoint framework [10] and an account on this experiment is given in [13]. To the best of our knowledge, process delegation and the integration of very small tools have not yet been explored.

The advantage that we see for the use of very small tools is that, from a process programming point of view, all established techniques remain applicable and tool invocation, e.g. through envelopes, can be handled in the same way as if it were more coarse-grained tools. The trade-off, however, is that tool integration becomes much more complex since a much greater number of tools have to be integrated. Also, we have doubts whether an acceptable tool performance is achievable in this approach.

This disadvantage is resolved if tools remain coarse-grain, but offer fine-grain services and notify events. However, invocation of services can no longer be achieved with enveloping, but more complicated techniques have to be used to achieve inter-process communication between tools and process engine. This complicates both the architecture of the tool and that of the process engine, since both have to be multi-threaded. Moreover appropriate language primitives for invoking services and receiving events have to be available in the process programming language. A further disadvantage, which holds for both fine-grained and process-centred tools, is that a centralised process engine does not scale to big software processes. It will become a performance bottleneck as soon as a large number of tools have to be controlled.

Completely decentralised process enactment, as in Viewpoints, solves this problem. In addition, developers can model and change their own process in a complete autonomous fashion. A major disadvantage of this approach, however, is that the overall progress of a process is neither modelled nor can it be monitored during enactment.

Process delegation seems very promising to us, because it overcomes this problem. A central process engine could remain in control to take care of the overall process progress, though also a certain degree of decentralisation and autonomy is achieved if fragments of the process are delegated to the process engine. However, a number of challenges arise, which to date we do not know how to meet. Process engine and tools need to agree on a protocol for downloading process fragments. Moreover the implications for process dynamics are unclear.

3 Process-Centred Tools

A way ahead that has been investigated in detail is the systematic construction of *process-centred tools* [6]. A process-centred tool offers all the services that are required by a particular process program and informs the program about the occurrence of relevant events. The rapid construction of these tools and their customisation towards particular process programs is accomplished by the *GOOD-STEP tool specification language* (GTSL) [7] and GENESIS, a compiler for this language that generates the desired process-centred tools. GTSL was used in a case study to generate a number of process-centred tools for use within the British Airways PSDE [8]. These process-centred tools were integrated with the SPADE environment using the SPADE Communication Interface [1]. A process program, defined in SLANG, implemented the C++ class library development and maintenance process of British Airways and invoked services from BA PSDE tools, which notified the process model of relevant events. An evaluation of this approach is given in [9].

GTSL is an object-oriented language, centred around a pre-defined library of classes. This library defines generic and reusable concepts to be supported by all tools. The language can then be used to add tool-specific concepts. This paradigm is also applied to services and events. GTSL supports a number of *generic services*, such as `create` (create a document with a root version), `edit` (open a particular version of a document for editing), `view` (open a document version for viewing), `derive` (derive a new version of a document) and numerous others. Likewise, *generic events*, such as `modify` (a document version has been modified by a particular user), are defined. Any tool specified in GTSL supports these generic services and events.

A number of dedicated language primitives are available to define *tool-specific services* and *tool-specific events*. They are declared in a *tool configuration* that serves as an interface definition for the specific services and events supported by a tool. An excerpt of such a tool configuration is displayed below:

```
CONFIGURATION ENTITY
  FROM CONFIGURATION GOAL IMPORT Goal;
  RELATIONSHIPS
    UsedIn:SET OF Goal;
  END RELATIONSHIPS
  SERVICES
    SERVICE $DefineName(n:STRING); // change name to 'n' ...
  END SERVICES;
  EVENTS
    EVENT $OpenGoal(g:Goal); // Open goal g  ...
  END EVENTS;
END CONFIGURATION ENTITY.
```

The tool could be part of an environment for goal-oriented KAOS requirement specifications [5]. Tool configurations declares relationships between documents, an example of which is the `UsedIn` relationship that relates an entity to the set of goals that use the entity. A service `DefineName` is defined that enables the process program to change the entity's name. The program may request the service to establish the name after a new entity definition document has been created. The event `OpenGoal` notifies the process engine that the user seeks to browse through relationship `UsedIn` to goal `g`.

The GOODSTEP case study at British Airways has demonstrated that it is feasible to construct and customise environments in a relatively short period of time. We believe that flexible tool construction facilities will become part of the CASE market. In fact, the different variants of the StP environment for OMT and Structured Analysis that have recently become available are customised versions of a common kernel, called StP/Core. StP/Core has an interpreter for QRL, a query and reporting language that facilitates queries and updates in StP's document repository. StP/Core and QRL also provide facilities for an integration with Sun ToolTalk so that tool builders can customise an StP environment for a specific process by adding tool specific fine-grained services in QRL. QRL, however, is a rather low-level language and is quite difficult to use by comparison with the high-level concepts available in GTSL.

4 Summary and Further Work

We have argued the need for process engines and tools to interface at a finer level of granularity. With currently available off-the-self tools this can only be achieved with difficulty. We have presented four architectural patterns for how fine-grained process guidance can be achieved, and have assessed these patterns. Of these the construction or customisation of process-sensitive tools with a tool specification language was discussed in detail.

A lesson that we have learned from the experiment in GOODSTEP is that there is a bounding problem: *should a process fragment be implemented by a process program or a process-centred tool?* In the above example, determining access rights could also be achieved by tools. The process engine would then have to inform tools whenever owners of documents were changed. Tools could then

store up-to-date ownership information of documents. To determine the access rights of a particular user to a document the tool could then check whether he or she was the owner and only then provide the full tool functionality, otherwise it might restrict the applicable functionality to read-only operations. A general solution to this bounding problem, however, is not clear to us at present.

The implementation of services in GOODSTEP was achieved on the basis of the message-based communication mechanism offered by Sun ToolTalk. It seems that, with OMG/CORBA [14] implementations becoming widely available now, there are better options than ToolTalk. CORBA seems preferable because it allows for a heterogeneous implementation and supports synchronous communication more naturally than ToolTalk. The way in which an object request broker can be incorporated into the tool architecture that we suggested in [6], however, needs further exploration.

Acknowledgements We are indebted to Mike Wagner who contributed the implementation of events and services to GTSL. Moreover we thank Sergio Bandinelli, Luigi Lavazza and Alfonso Fuggetta for the fruitful discussions and cooperation we had on the matter of integrating process-centred tools into the BA PSDE. The comments from Olly Gotel helped to improve the presentation of this paper.

References

1. S. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza. The Architecture of the SPADE-1 PSEE. In B. Warboys, editor, *Proc. of the* 3^{rd} *European Workshop on Software Process Technology, Villard de Lans, France*, volume 772 of *Lecture Notes in Computer Science*, pages 15–30. Springer, 1994.
2. S. Bandinelli, A. Fuggetta, and C. Ghezzi. Process Model Evolution in the SPADE Environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, 1993.
3. S. Bandinelli, E. Di Nitto, and A. Fugetta. Supporting cooperation in software development. Technical Report 32-95, Dipartimento di Elettronica ed Informazione, Politecnico di Milano, June 1995. Submitted for Publication.
4. M. R. Cagan. The HP SoftBench Environment: An Architecture for a New Generation of Software Tools. *Hewlett-Packard Journal*, 41(3):36–47, June 1990.
5. A. Dardenne, A. van Lamswerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20:3–50, 1993.
6. W. Emmerich. *Tool Construction for process-centred Software Development Environments based on Object Database Systems.* PhD thesis, University of Paderborn, Germany, 1995.
7. W. Emmerich. Tool Specification with GTSL. In *Proc. of the* 8^{th} *Int. Workshop on Software Specification and Design, Schloss Velen, Germany*, pages 26–35. IEEE Computer Society Press, 1996.
8. W. Emmerich, J. Arlow, J. Madec, and M. Phoenix. Tool Construction for the British Airways SEE with the O_2 ODBMS. Technical report, City University London, Dept. of Computer Science, 1996. Submitted for Publication.

9. W. Emmerich, S. Bandinelli, L. Lavazza, and J. Arlow. Fine grained Process Modelling: An Experiment at British Airways. In *Proc. of the* 4^{th} *Int. Conf. on the Software Process, Brighton, United Kingdom*. IEEE Computer Society Press, 1996. To appear.
10. A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: a framework for integrating multiple perspectives in system development. *Int. Journal of Software Engineering and Knowledge Engineering*, 2(1):21–58, 1992.
11. G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf. MERLIN: Supporting Cooperation in Software Development through a Knowlege-based Environment. In A. C. W. Finkelstein, J. Kramer, and B. Nuseibeh, editors, *Advances in Software Process Technology*, pages 103–129. Wiley, 1994.
12. G. E. Kaiser, P. H. Feiler, and S. S. Popovich. Intelligent Assistance for Software Development and Maintenance. *IEEE Software*, pages 40–49, May 1988.
13. B. Nuseibeh, A. Finkelstein, and J. Kramer. Fine-Grain Process Modelling. In *Proc. of the* 7^{th} *Int. Workshop on Software Specification and Design, Redondo Beach, California*, pages 42–46. IEEE Computer Society Press, 1993.
14. R. M Soley, editor. Object Management Architecture Guide. Technical report, Object Management Group, 492 Old Connecticut Path, Framingham, MA 01701, USA, 1992.
15. SunSoft. *ToolTalk 1.1.1 Reference Manual*. SunSoft, 2550 Garcia Avenue, Mountain View, CA 94043, USA, Solaris 2.3 edition, 1993.
16. S. M. Sutton, D. Heimbigner, and L. Osterweil. Language Constructs for Managing Change in Process-Centred Environments. *ACM SIGSOFT Software Engineering Notes*, 15(6):206–217, 1990. Proc. of the 4^{th} ACM SIGSOFT Symposium on Software Development Environments, Irvine, Cal.
17. G. Valetto and G. Kaiser. Enveloping "Persistent" Tools for a Process-Centred Environment. In W. Schäfer, editor, *Proc. of the 4th European Workshop on Software Process Technology, Nordwijkerhout, The Netherlands*, volume 913 of *Lecture Notes in Computer Science*, pages 200–204. Springer, 1995.
18. A. I. Wassermann and P. A. Pircher. A Graphical, Extensible Integrated Environment for Software Development. *ACM SIGPLAN Notices*, 22(1):131–142, 1987. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Palo Alto, Cal.

This article was processed using the LaTeX macro package with LLNCS style