Instances and Connectors: Issues for a Second Generation Process Language

B.C. Warboys[1], D. Balasubramaniam[2], R.M. Greenwood[1], G.N.C. Kirby[2], K. Mayes[1], R. Morrison[2], and D. Munro[2]

[1] Informatics Process Group (IPG) Department of Computer Science
University of Manchester, Oxford Road, Manchester, M13 9PL, UK
{brian,markg,ken}@cs.man.ac.uk
[2] School of Mathematical and Computational Sciences
University of St Andrews, North Haugh, St Andrews, Fife, KY16 9SS, UK
{dharini,graham,ron,dave}@dcs.st-and.ac.uk

Abstract. Over the past decade a variety of process languages have been defined, used and evaluated. It is now possible to consider second generation languages based on this experience [1, 3, 4]. Rather than develop a second generation wish list this position paper explores two issues: instances and connectors. Instances relate to the relationship between a process model as a description and the, possibly multiple, enacting instances which are created from it. Connectors refers to the issue of concurrency control and achieving a higher level of abstraction in how parts of a model interact. We believe that these issues are key to developing systems which can effectively support business processes, and that they have not received sufficient attention within the process modelling community. Through exploring these issues we also illustrate our approach to designing a second generation process language.[1]

1 Background

Over the past 8 years the Informatics Process Group at Manchester has developed a number of enactable process models using ICL's ProcessWise Integrator (PWI) [2] as our support technology, more recently linked with a Web interface in our ProcessWeb system [16, 6].

In parallel the research group has undertaken considerable work on developing a process modelling method which places process models at a pivotal postion between the business process and the IT systems providing appropriate support. Key to this is a clear mapping between models which can be discussed and verified with people in the business, and models which are used by a support system [8].

In our industrial case studies, the feedback has been that both descriptive models which allow people to discuss a process, and enactable models which support the people as they are doing the process, are valuable. However, the

[1] This work is supported by UK EPSRC grants GR/L34433 and GR/L32699

cost of developing an enactable model is seen as prohibitive, chiefly because of the "low level" coding skills involved. Better process languages are needed for process support systems to realise their potential.

2 Model Instances

One of the reasons for modelling and supporting a process is often the number of instances of that process with which the business deals. For example an insurance company may have one process instance for each insurance application being considered, one process instance for every insurance policy currently in force. However, the choice of a process instance is not trivial: alternatives include one process which deals with batches of applications, or one process per insured customer which deals with all their policies.

Process model instances are not completely independent. The normal situation is that people will be involved in more than one process and need an interface which allows them to switch between them. As processes often last for some time, people frequently wish to switch between understanding what the process is and actively playing their part within it. For them the relationship between a process instance and its description is important for the whole duration of the process. This "reflexive practice", thinking about the process while doing it, naturally leads to a desire to evolve the process as the details of the real world situation become clear.

It is also common to want a "higher-level" process which can receive feedback from others. Continuing the insurance company example there might be a requirement for a process which receives data from individual applications to track the company's overall exposure in specific sectors. Finally, process model instances are not independent because a business needs to learn from the experience of the past and present instances to improve the performance of future ones. It is therefore natural to see the process description, used to create instances, as itself being the output of a "meta-process".

3 Connectors

In modelling business processes, our emphasis is on the collaboration and coordination of people and the IT systems which they use. (This involves communication between users, between users and tools, and between tools.) In general, first generation process languages offer good abstractions for activity but not for communication. Pratten [7] has termed this the "wire syndrome", based on the observation that hardware designers also have good abstractions for functional blocks but not for the wires which connect them. A typical effect of this is a model which makes business executives worry about how their internal postal system works rather helping them to focus on the major issues.

The choice of appropriate connectors is key to achieving abstract models of interaction which enable people to effectively design and reason about processes. One of the major discontinuities between our user-centred models and enactable ones is that an abstract interaction such as "insurance expert and manager agree the price" must be implemented in terms of a complex set of possible message sequences. Understanding how these implement the required collaborative protocol between the parties involved can be very tricky. This makes the model hard to understand as it obscures the core process. It also makes the model hard to change, and hard to break into potentially reusable parts.

The term "connectors" comes from work in software architecture [11]. It is now widely recognised that an effective architecture needs to deal as much with connectors as the components being connected. This enables the coordination and interaction aspects of the architecture to be separated from the computational ones [10]. This separation of computation and coordination is also a promising way of achieving reuse and composability within process models. Alternative approaches to providing explicit support for collaboration, such as COO [5], concentrate on activities: do activities need to be serialized, what is the appropriate visibility for intermediate results.

4 Old and New Process Languages

The ProcessWise Integrator PML (PWI PML) was designed during the IPSE 2.5 [15] project and has changed little since 1989. In PWI PML a process is represented as set of role instances connected by interactions. Each role instance is a separate thread of control with its own local data, and set of actions (activities). The role instances communicate through interactions, which are typed, asynchronous, buffered channels. Interactions are also used for user and tool communication through specialised "user agent" and "tool agent" roles.

There is a pre-defined action "StartRole" which creates a role instance from its class name and a set of class definitions. A role instance can be provided with some initialisation parameters, including references to interactions which enables it to communicate with other roles. It is quite common for a PWI PML program to have a main or setup role whose function is to create the appropriate network of role instances and interactions.

A set of class definitions exist as a type within the language and this is used to provide an incremental compilation facility, where PWI PML code as a string is compiled in the context of an existing set of class definitions. It is possible to extract a class definition as a string but this is a rudimentary interface added for diagnostic purposes.

PWI PML interactions provide a basic facility for sending typed messages between roles. Since references to interactions are included in the types which can be communicated, it is possible to develop systems where there is a dynamic set of connections between role instances. Hovever, if the interactions are supporting a higher-level communication between role instances, such as a long transaction, then the details of this are embedded in the roles and difficult to separate from the other computation involved. Our experience is that it is not unusual for almost half the PWI PML code for a model to be dealing with general concurrecy issues rather than the specifics of the process involved.

Recently Sutton and Osterweil have been working on a second generation process language [14], based on their extensive experience using APPL/A. They identify a number of issues which they believe are key to second generation process languages: semantic richness, ease of use, appropriate abstractions, process composability, visualisation, and multiple paradigms. In their language JIL, the process step is proposed as a key abstaction. One of the key features of steps is that they can have a factored description enabling different modelling styles to be freely combined.

JIL is a language designed with software process modelling in mind, in contrast to our emphasis on business processes in general. The issue of relating a process to its instances is not given any particular emphasis, although there are some suggestions about process visualisation. Within the step abstraction, there is a facility for concurrent sub-steps, and [13] details various means for specifying parallel control flow. However the emphasis is on detecting and avoiding unwanted interference between parallel activities, rather than supporting interaction between them.

5 A Promising Approach to Model Instances

The importance of model descriptions and instances tends to suggest a reflexive language. One in which a process instance (process as code) can refer to its own description (process as data), and where additional description (data) can be incorporated in an existing instance (code). In addition, we may want library facilities of both types: model descriptions which can be incorporated in other models as they are instantiated, instances which provide part of the appropriate environment for a new process. The language must also enable us to descripe meta-processes which themselves instantiate, monitor and change process instances, and meta processes which update the libraries.

Hyper-programming [9] is a programming style associated with persistent programming systems. In persistent programming systems both source and object code can exist as persistent objects. In a normal program, when the programmer wishes to refer to some existing object, which could be code or data, a textual description which describes how to locate the required object is used. In a persistent programming system the object is often already available at the time when the program is composed. In these situations hyper-programming allows a persistent link to the object rather than a textual description to be used. By analogy with hyper-text, this style of programming containing both text and links to objects is called a hyper-program.

In [9] hyper-programming is applied in the context of a software development environment, to link versioned source and object code together. As the links are persistent, real rather than intended configurations can be discovered; developers can browse the store and understand the current situation. We propose to use the same approach to link process descriptions and instances and so support "reflexive practice".

A natural progression is that some processes will not be completely described ab-initio, part of the process will initially exist as a description which is transformed and instantiated as the process progresses. This could be particularly significant for processes, including meta-processes, which last for a long time and cannot be predicted with any degree of certainty.

6 Getting to grips with connectors

With connectors we want to abstract above the level of message passing. In [7] the key concept is one of shared behaviour. This indicates that two or more concurrent elements interact, that is they depend on each other in some way, without attempting to give the details of how this is achieved. One way of making this concrete is to represent such shared behaviour as shared data along with an agreed protocol for accessing it. This includes basic message passing as a degenerate case; one party writes to the shared message and then the other reads it.

Communicating Actions Control System (CACS) [12] is an abstract operational model for specifying flexible concurrency control schemes. Its goal is to allow computations to control the coherence of their operations on shared data in an understandable and flexible manner. In a CACS environment computations which share data are annotated with markers which specify how the data sharing is to be controlled. The represents the starting point for our approach to the support for concurrency and collaboration in a second generation process language. The hope is that this will lead to significant reuse of concurrency schemes across different models. The lack of this type of reuse, in practice, was one disappointing aspect of our experience with existing PMLs.

7 Conclusions

Our view is that businesses should not view an enactable process model simply as a way of achieving process conformance. Process models, including enactable ones, should be valued assets which help a business to learn about its current situation and give careful consideration to its future improvement.

Our approach is to have a small core language augmented with packages. A hyper-programming package which provides support for the linking of model descriptions and instances is one key package. A cooperation control package, based on CACS, is another. This small core plus packages approach is designed for flexibility, and we expect to have support in the system architecture for incorporating different packages. This approach reflects how much less we know about what will make a successful process language than we did 10 years ago.

References

1. V. Ambriola, R. Conradi, and A. Fuggetta.: Assessing Process-Centered Software Engineering Environments. *ACM Transactions on Software Engineering and Methodology*, 6(3):283–328, July (1997).

2. R.F. Bruynooghe, R.M. Greenwood, I. Robertson, J. Sa, R.A. Snowdon, and B.C. Warboys.: PADM: Towards a total process modelling system. In A. Finklestein, J. Kramer, and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pages 293–334. Research Studies Press, (1994).
3. B. Curtis, M.I. Kellner, and J. Over.: Process modelling. *Communications of the ACM*, 35(9):75–90, September (1992).
4. A. Finklestein, J. Kramer, and B. Nuseibeh, editors.: *Software Process Modelling and Technology*. Research Studies Press, (1994).
5. C. Godart and D. Dietrich.: Stepwise specification of interactive processes in COO. In Wilhelm Schäfer, editor, *Software Process Technology - Proceedings of the* 4^{th} *European Workshop*, pages 220–239, Noordwijkerhout, Netherlands, April (1995). Springer-Verlag. Lecture Notes in Computer Science 913.
6. R.M. Greenwood and B.C. Warboys.: ProcessWeb - Process Support for the World Wide Web. In Carlo Montangero, editor, *Software Process Technology - Proceedings of the* 5^{th} *European Workshop*, pages 82–85, Nancy, France, October (1996). Springer-Verlag. Lecture Notes in Computer Science 1149.
7. P. Henderson and G.D. Pratten.: POSD - A notation for presenting complex systems of processes. In *Proceedings of the 1st IEEE International Conference on Engineering Complex Computer Systems*, (1995). URL http://www.ecs.soton.ac.uk/~ph/Papers.htm.
8. P. Kawalek.: *A Method for Designing the Software Support of Coordination*. PhD thesis, University of Manchester, December (1996).
9. R. Morrison, R.C.H. Connor, Q.I. Cutts, V.S. Dunstan, and G.N.C. Kirby.: Exploiting Persistent Linkage in Software Engineering Environment. *The Computer Journal*, 38:1–16, (1995).
10. D.E. Perry.: Directions in process Technology - an architectural perspective. International Workshop on Research Directions in Process Technology, Nancy, France, (1997).
11. M. Shaw and D. Garlan.: *Software Architecture: perspectives on an emerging discipline*. Prentice-Hall, (1996).
12. D. Stemple and R. Morrison.: Specifying Flexible Concurrency Control Schemes: An Abstract Operational Approach. In *Proceedings of* 15^{th} *Australian Computer Science Conference*, pages 873–891, Hobart, Tasmania, (1992).
13. S.M. Sutton Jr. and L.J. Osterweil.: Programming Parallel Workflows in JIL. University of Massachusetts, April (1997).
14. S.M. Sutton Jr. and L.J. Osterweil.: The Design of a Next-Generation Process Language. Technical Report CMPSCI Technical Report 96-30, University of Massachusetts, (1997). in Proceeding of the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich (LNCS 1301).
15. B.C. Warboys.: The IPSE 2.5 project: Process modelling as the basis for a support environment. In *Proceedings of the First International Conference on Software Development, Environments and Factories*, Berlin, (1989). Pitman Publishing.
16. B. Yeomans.: Enhancing the World Wide Web. student project report, (1996).