The Oikos Services for Object Management in the Software Process*

Vincenzo Ambriola, Giovanni A. Cignoni and Carlo Montangero
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy

Abstract. A software development process is strictly related to the representation and the management of the involved objects. Software products, tools, and computational resources are typical objects. It is convenient to distinguish the definition of software process activities and the issues that pertain to object management. Standard services have been introduced in Oikos to provide process activities with a set of primitive functionalities for object management. These standard services present the Object Management System functionalities at an abstraction level that well matches the definition of the process activities.

1 Introduction

A *software development process* is usually described as a set of concurrent activities [12, 13]; its *performance* is the interpretation of these activities by human people and machines [18]. Each activity has an associated set of *agents* and a set of *objects*. The first set consists of the agents in the software process: automatic components of the process and people participating to its performance. The set of objects includes development products, tools in use, computational resources, and the representation of the components of the process themselves. For instance, the objects of an edit-compile cycle are the modules under development and the tools in use; the objects related to a task assignment are the representations of the task and of the technician who will carry it out.

In this perspective it is important to define a model that takes into account the representation and the management of all the objects involved in the software process. Such a model must allow a clear distinction among the peculiarities of object management and the description of process activities. In other words, the model must introduce a set of primitive operations on objects, thereby defining an abstraction layer on top of which the activities of the software process can be defined.

In Oikos, a software process is described as a set of co-operative *entities*. The idea of defining a number of basic concepts for the software process is currently widely accepted [13]: a specific software process is thus obtained by specialising and

* This work is funded by Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo del CNR, MURST 60%, and ESPRIT BRA Promoter WG.

composing these concepts. In Oikos the following entities are introduced, one for each software process concept: *processes*, *environments*, *offices*, *desks*, and *services* [1]. Typically an entity is defined in terms of simpler entities according to a composition strategy that introduces several abstraction levels [2]. The leaves of the resulting hierarchy are simple entities, ie entities that are not decomposed and that constitute the base of the software process description. Among these simple entities, services are introduced to manage the resources exploited in the software development. Resources are naturally grouped in classes and a different service is introduced for each class. Moreover, each service defines the interface used by software process activities to access the resources, ie the objects.

The need of encapsulating object management functionalities in a well identified component of the run time support is commonly addressed in the literature related to software process representation and enactment. For this purpose, Arcadia [24] and Odin [10] introduce *object managers*; EPOS [11], Marvel [17] and Spade [4] exploit *databases*; Merlin [20] relies on a *knowledge base*. In Oikos, objects are managed by an *Object Management System* in a peculiar way: management functionalities are indirectly presented to software process activities by services. The major advantage of this approach is that it leads naturally to a modular definition of the specific resource model of a given process.

The next section describes object management from the viewpoint of the software process introducing the concept of Oikos service. Section 3 presents the core of the model for the management of products and tools, ie the basic objects and the operations shared by any process enacted in Oikos. Section 4 describes the architecture of the Oikos OMS and Section 5 points to the evolution of the current implementation.

2 The Service Concept

The idea of services as resource managers is largely used in concurrent programming and operating systems. Similar concepts are found in *monitors* [15] and *client-server* architectures [23]. The main objective in applying these concepts to process-centred development environments is to achieve a better structure and more modularity both in the process description and in the architecture of its enactment support.

In this context, a *resource* is any element needed to the software process for which the notions of access, sharing, and arbitration are meaningful. The service monitors the accesses to a shared resource and solves the related concurrency conflicts. Different classes of resources are managed by different services each one defining the relevant access protocols.

A *protocol* defines an access mode to a resource of a given class: it defines a set of functionalities that act on the resource and the policies that rule their invocation. A service is characterised by the whole set of functionalities offered by its protocols. As an example, consider a service that manages a resource consisting of a document repository: its functionalities include creation and deletion of a document, and access controlled by read or write rights. The service protocols define the available functionalities in terms of the resource state. For instance, a protocol will control

concurrent write accesses to a document and the rights to deposit or delete a document.

Fig. 1. Relations between *clients*, *sessions*, and *services.*

The notion of service is general and each specific service depends on the class of the managed resource. A service *instance* manages a specific, concrete resource of the class for which the service has been defined. *Clients* may use the functionalities offered by a service instance via a mechanism based on *sessions*. A client issues a request to open a private session; the client can then use the service functionalities until the session is closed either by an explicit request or because of a service decision. Each session has an associated protocol; only one protocol can be associated to a session. Clients and service instances interact only via sessions. A client can request multiple private sessions with each service instance: more than one session can be simultaneously open, even with different protocols for the same client.

The request to open a session is a functionality common to all services. Like any other functionality, this request must follow a *login* protocol associated to a login session. The login session is a public session automatically opened when a service instance is created and available to its clients as long as the service instance is active.

In Oikos, a service instance is meaningful only as a component of a more complex entity [1]. Each service instance is *activated* as a component of a unique

entity, called its *mother*. On the other side, a service instance is *available* to its mother and to all the entities in the hierarchy rooted in the mother. This set of entities constitutes *the set of clients* of the service instance. Access to a service instance is granted to its clients by the login session. Availability does not necessarily mean that a session request will be satisfied: in general the service makes a decision according to its state.

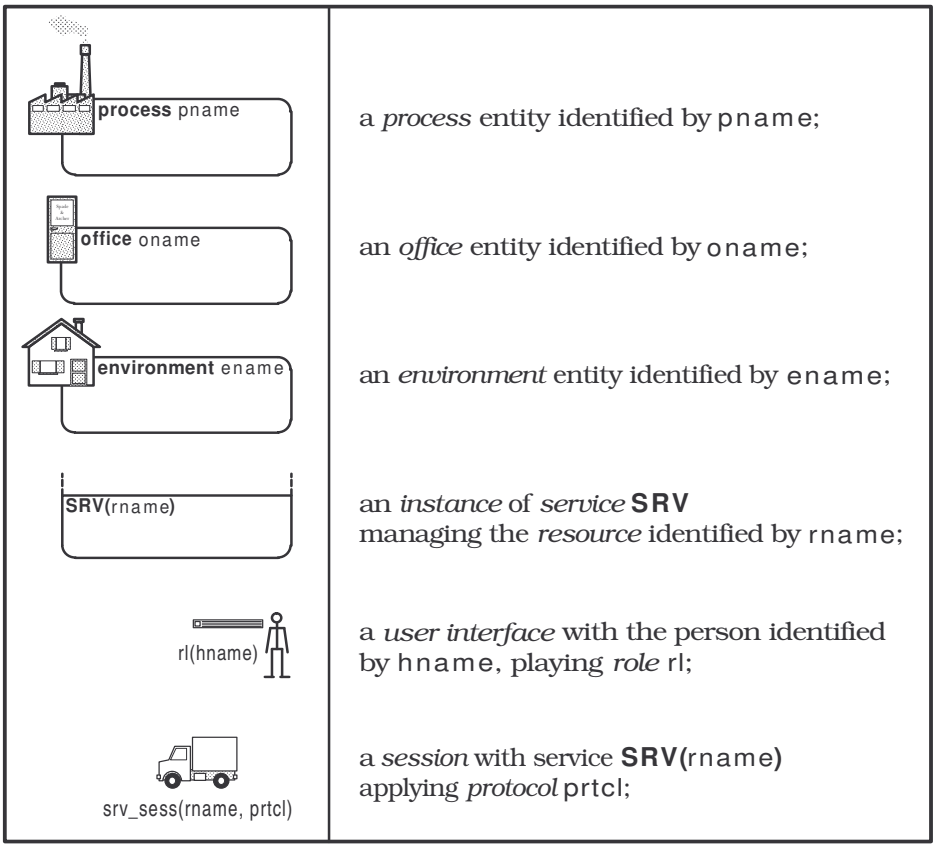

Table 1. Graphical representation of software process components.

Figure 1 sketches the relations between clients, sessions, services, and the managed resources. The graphical notation is explained in Table 1. Figure 2 graphically presents an example of the use of services in a simple software process. The whole process corresponds to the composed entity named development. This entity consists of the manager office and of several environments. A developer performs his task in each environment. Moreover, process development consists of two service instances, namely PRS(main), as an instance of the *Product Repository Server* (PRS), and TRS(main), as an instance of the *Tool Repository Server* (TRS). Since they have been activated in the external entity they are also available to the internal components, ie to the office manage and to the environments develop_i.

Another service, the *Workspace Server* (WS), is also used in the process. Its instances are labelled with the name of the managed workspaces: WS(manage) in the office manage, and WS(develop_i) in each develop_i environment. The manager accesses WS by session ws_sess(manage, access) and PRS functionalities by session prs_sess(main, management). The arguments in the session name define the resource and the protocol. The manager can deposit development products in the product repository main since the *management* protocol allows him to do so. Each developer has a session with each of the service instances WS(develop_i), TRS(main), and PRS(main). Session prs_sess(main, access) allows the developer, via the *access* protocol, to check-out and check-in the products belonging to the main product repository. The same protocol forbids the developer to destroy or deposit products, leaving this responsibility to the manager.

The nature of the sample resources, product repository, tool repository, and workspaces will be discussed in detail in the next section.

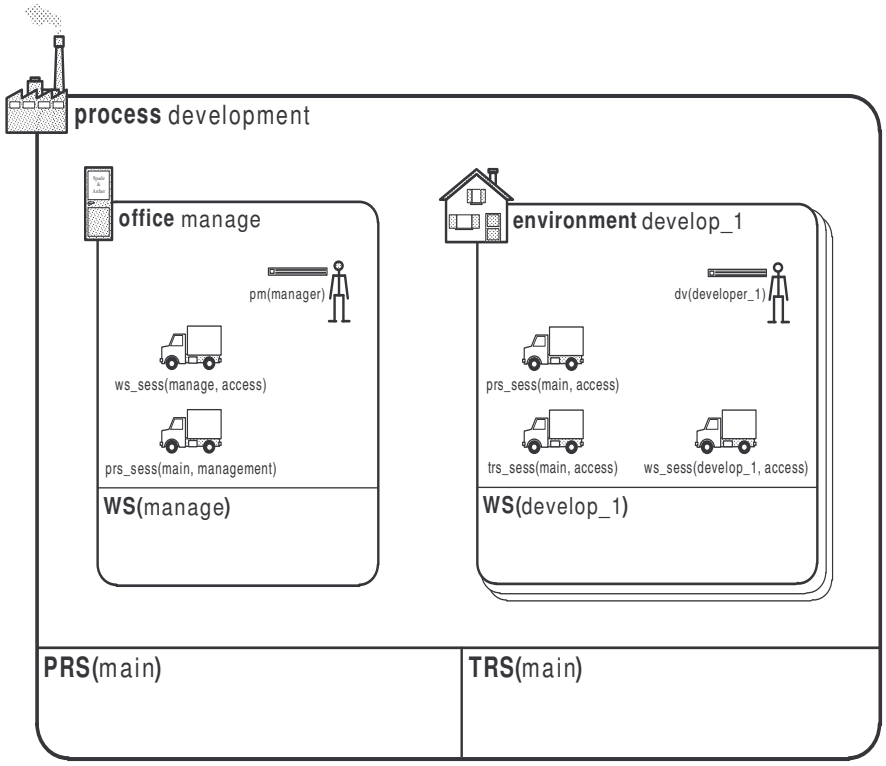

Fig. 2. Representation of a software process in Oikos.

3 Modelling Products and Tools

Most policies for the management of objects like products and tools are common to all software processes. In Oikos, in order to foster reuse and standardisation, a set of *standard services* are predefined to offer functionalities to access the process objects. Among these, the already mentioned standard services PRS, TRS, and WS offer access protocols that guarantee consistent access with respect to the basic model for products and tool management. Each standard service supports the management of a specific class of objects or of a facet of the model: for instance, service WS allows tthe application of tools to products.

The main characteristic of the product model in Oikos is the distinction between *deposited documents* and *working documents*. Deposited documents are the real products of the software process, ie those documents that in their development have reached a consistent state, even if not necessarily the ultimate one. A *product repository*, ie the resource managed by an instance of PRS, holds a set of deposited documents. The protocols of service PRS define the concurrency control of the access to deposited documents, according to a policy based on the *check-out/check-in* mechanism: a deposited document can be locked, therefore ensuring that it can be modified in just a workspace at a time. Moreover, it is possible to constrain access to selected documents exploiting access keys. Documents held in repositories may have *revisions* and *releases* according to the SCCS/RCS model for version control [21, 25].

Many approaches to software process modelling provide a specific representation of the environment where documents are operated on. For instance, Adele2 [6] has the notion of *working environment* and Merlin [22] that of *working context*. In Oikos a *workspace* is a set of computational and memory resources allocated to a given task to hold the products currently under development, namely the *working documents*, and the tools to work on them. Standard service WS defines the access protocols to a workspace, ie how it is possible to manage working documents and to apply tools to them.

Finally, both deposited and working documents have a type used to control tool application. The set of document types of a given software process can be partially ordered in such a way that tools can be applied to documents of different types in a controlled way. For instance, given the partial order shown in Figure 3, and given that an editor is applicable to documents of type *text*, the same editor can be applied also to documents of types *source*, *c_module*, and so on.

The tool model has three main facets: tool *representation*, *installation*, and *application*. A *signature* is associated to each tool. The signature specifies the number and the types of input working documents, the number and the types of output working documents, and the parameters for tool application. As shown in Figure 4, the output working documents are grouped in two classes: those that are created by the tool application and that are not in the workspace before, and those modified which are a subset of the input documents already in the workspace and that are changed by the tool application.

A *tool repository*, ie the resource managed by an instance of TRS, holds a set of signatures. The protocols of service TRS have the functionalities to modify the set of signatures in the resource and to install the tools in a workspace. Installing a tool in a workspace amounts to defining its application environment, ie the set of documents

on which the tool can work and the physical resources used by the tool (the machine that will run it and the X display that will be used to interact with the user, whenever the tool needs one).

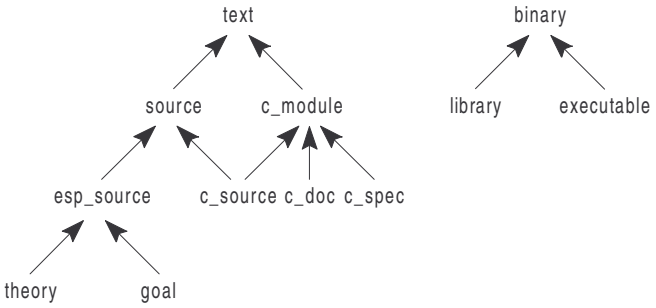

Fig. 3. An example of document types ordering.

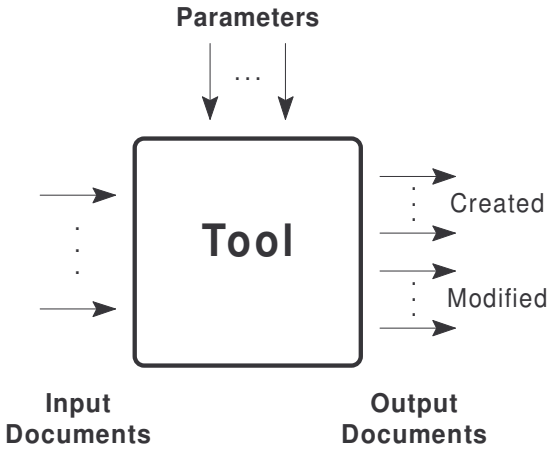

Fig. 4. Tool signature.

Finally, the details of tool application are defined in the tool model. There are three phases:

- ***check***: the correctness of the application request is verified with respect to the tool signature, the availability of the tool and of the input working documents in the workspace;
- ***execution***: the tool is run on the resources associated to the workspace;
- ***update***: the set of working documents in the workspace where the tool has been applied is updated according to the outcome of the execution.

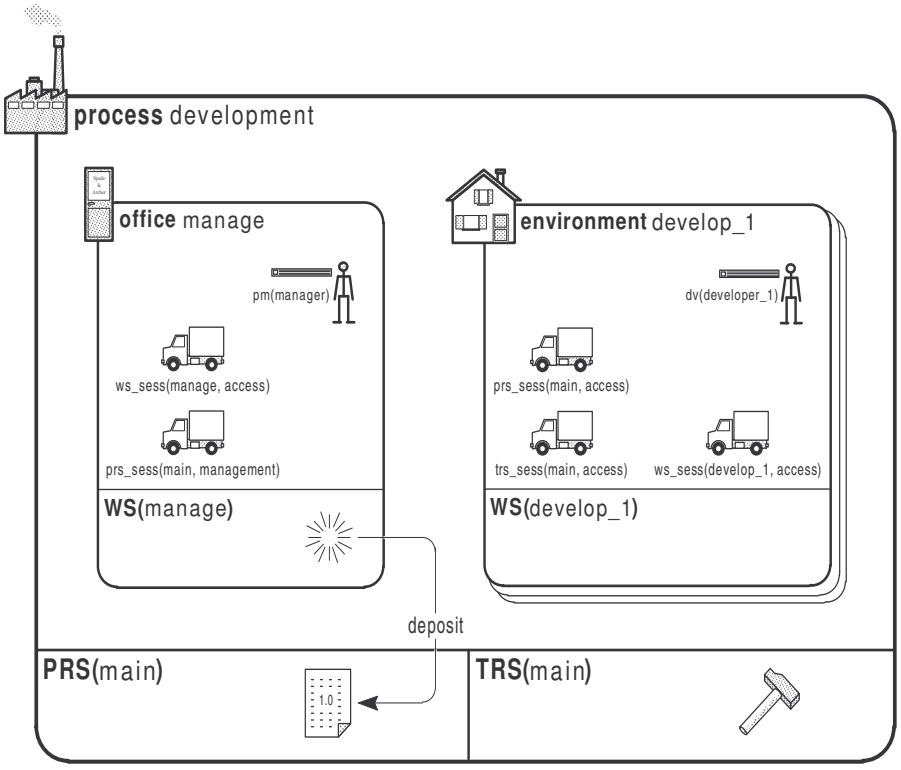

Fig. 5. *Deposit* of a document.

A fourth important facet of tool management is related to tool integration. Oikos fulfills the requirements of *Rapid Tool Integration* [19]: any tool which is considered useful to the software development can be easily integrated, without the need of modifying the tool. The approach is the same of Marvel [14], where tools are wrapped in an *envelope* that redefines the tool interface to suit the environment needs [3].

As an example of use of standard services in object management we describe a fragment of the life of a source module in the process introduced in Figure 2. Figure 5 shows how the first version of the source, which has been created by the manager in his own workspace, is deposited in the product repository associated to the PRS(main). The document is deposited via session prs_sess(main,

management). This deposited document then makes the source available to all the developers involved in the process.

Figure 6 shows the state of the process after a developer has copied the source module, as a working document, in his own workspace. The copy is done by a check-out operation via session prs_sess(main, access). After this check-out operation, the document in the repository is locked so that no other developer can modify it. Moreover, developer dv(developer_1) accessed the tool repository managed by TRS(main) and installed the needed tools, eg an editor and a compiler.

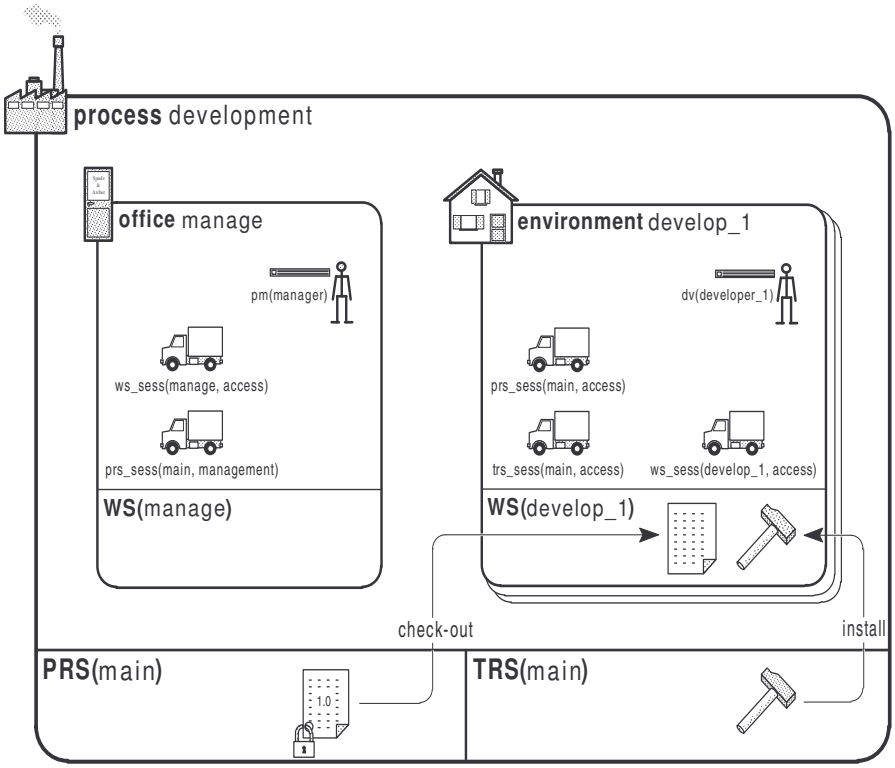

Fig. 6. *Check-out* of a document and *install* of a tool.

The developer then applies the tools to the working document in the workspace via session ws_sess(develop_1, access). Finally, once the module has been completely coded, the developer will check-in the document, always via session prs_sess(main, access): a new version has been created and the lock on the previous one has been released, as shown in Figure 7.

Different typical development policies, with respect to tool access in the development context and to documents visibility at different stages of the process, can be represented in Oikos by suitable tailoring of service instances availability to the process entities.

4 OMS Architecture

Standard services are responsible for the presentation of the operations on objects to the software process. Besides presentation, object representation must also be considered, as well as the management of their physical components. Several approaches consider an object as a set of *attributes* and a *contents* [5, 7]. The set of attributes characterises the object representation, ie the form of its presentation to the software process; the contents correspond to the objects substance, ie its physical nature as it exists in the real world of the development. Accordingly, the Oikos OMS is structured in three abstraction levels, as depicted in Figure 8.

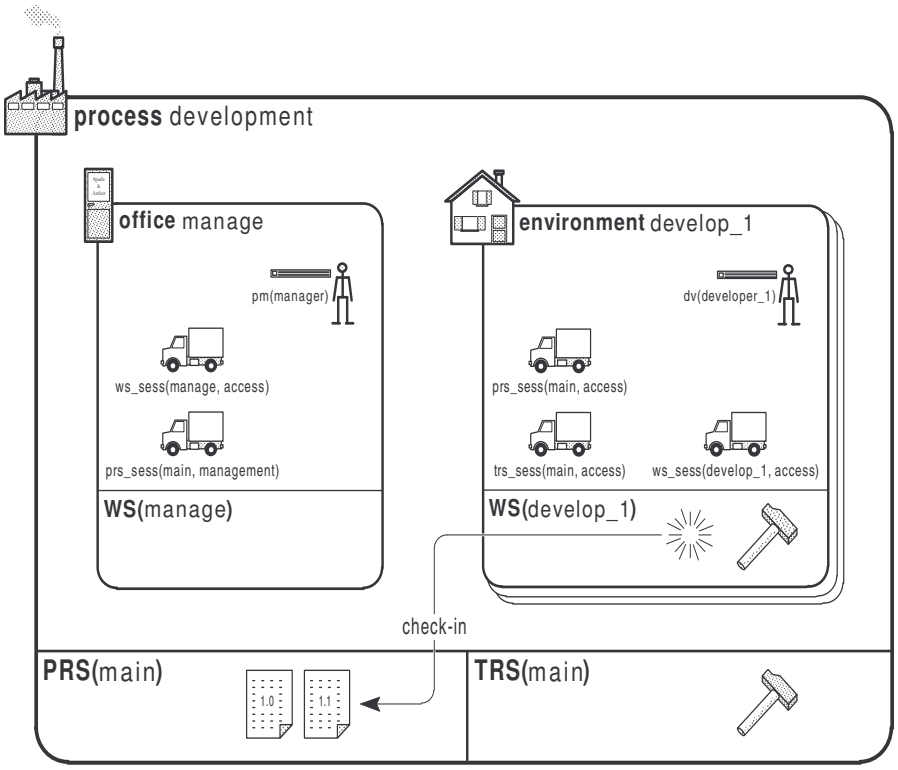

Fig. 7. *Check-in* of a document and creation of a new version.

- ***Service level***: this level is in charge of object presentation to the software process entities. The Oikos standard services define the protocols used by the entities to access the objects under development. The services, being entities themselves, belong naturally to the description of the software process: in fact they provide the basic functionalities in the description of the process. To this purpose, the interfaces of the services toward the process are defined in Patè, the Oikos enactment language.

- ***Object Level***: this level is in charge of the management of the object form, ie of the objects representation with respect to the process description which has the property of being independent of its physical nature. Goals of this level are the persistency of the object representation and the availability of a logical global view of the objects. In this way, it is possible to manage the relations among different classes of objects such as the link between a working document and the original deposited document in a repository or the link between a tool installed in a workspace and the tool signature in the tool repository. This global view of the objects defines an essential part of the process state upon which it is possible to make queries and verify properties.
- ***Item Level***: this level is in charge of the substance of the objects. Most classes of objects represented at the object level have a corresponding substance in the real world: the item level interfaces the OMS to the operating system supporting the Oikos environment. Therefore the item level represents also the portability level of Oikos.

In the OMS implementation, the above level are layered virtual machines each devoted to support a specific facet of the object nature. The object level plays a central role, while the other two are interfaces.

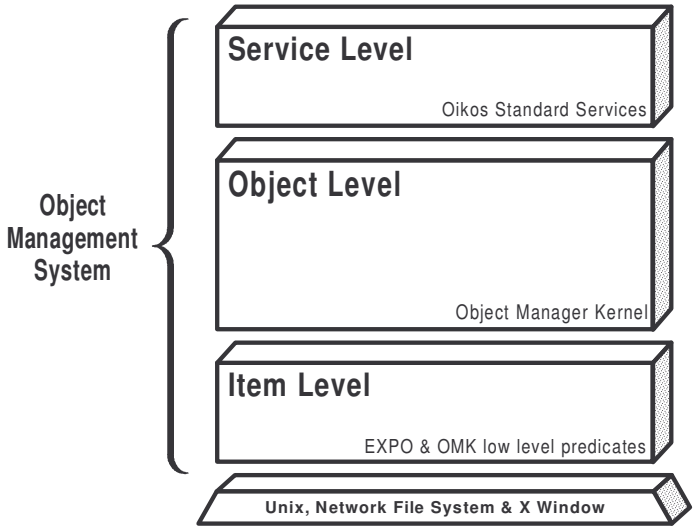

Fig. 8. Structure of the Oikos OMS.

The *Object Manager Kernel* (OMK) is the OMS component that implements the functionalities of the object level. The OMK manages the set of resources, ie repositories and workspaces. It supports resource allocation and deallocation and implements the operations on objects. Currently the OMK is implemented as a logic database defined in LDL [9]. Operations on objects are implemented as transactions on the database.

With respect to the integration of the OMK and the Item Level we adopted an approach like the one introduced in [16]: the Item Level consists of several components each one in charge of the management of a specific class of objects. This approach naturally matches the diversity of the physical objects to be considered: the OMK is then viewed as a means to integrate specialised lower level managers. As a consequence, there is a great flexibility to choose the mechanism for the physical object management at this level. The current implementation interfaces the underlying file system via a set of external predicates written in C and imported in LDL. On the other hand, tool integration and execution exploit the functionalities offered to this purpose by EXPO [3].

5 Future Development

A first foreseen extension of the OMS is related to the process representation and its enactment state. Two new standard services, *Process Model Server* and *Process State Server*, are being designed together with the definition of the relevant object classes, namely the objects capturing the entities in the process model and their states.

A number of operations related to the management of computation resources and of the envelopes to integrate tools are now performed, at low level, outside Oikos. It is foreseen that these operations will be presented to the software process entities via new protocols of the relevant standard services, therefore providing a more uniform environment for process enactment.

An independent extension is related to the document model as a whole. The software process definer needs to introduce structured types in order to tailor the document model to a specific software process. This entails the introduction of a more sophisticated approach to concurrency control in the access to documents and to the related merging and versioning policies.

Acknowledgements

We want to thank our users Cristina Iorio, Silvana Fantoni, Alessandra Tesauro, and Stefano Tirabassi for their constructive criticisms. Tito Flagella and Mauro Gaspari developed the first version of the Oikos Run-time Support.

References

1. V. Ambriola, C. Montangero, “Hierarchical Specification of Software Processes in Oikos”, Proceedings of the 7th International Software Process Workshop, Yountsville, October 1991. 2. V. Ambriola, C. Montangero, “Oikos at the age of three”, Proceedings of the 2nd European Workshop on Software Process Technology, Trondheim, 1992. 3. V. Ambriola, C. Montangero, M. Gaspari, T. Flagella, “EXPO: a Framework for Process Centered Environments”, Technical Report 21/92, Dipartimento di Informatica, Università di Pisa, August 1992. 4. S. Bandinelli, A. Fuggetta, C. Ghezzi, S. Grigolli, “Process Enactment in SPADE”, Proceedings of the 2nd European Workshop on Software Process Technology, Trondheim,

September 1992. 5. P. Baumann, D. Köhler, “Archiving Versions and Configurations in a Database System for System Engineering Environments”, Proceedings of the International Workshop on Software Version and Configuration Control, Grassau, January 1988. 6. N. Belkatir, J. Estublier, W. Melo, “A Support to Large Software Development Process”, Proceedings of the First International Conference on the Software Process, Redondo Beach, June 1991. 7. L. Bendix, “Automatic Configuration Management in a General Object Based Environment”, Proceedings of the 4th International Conference on Software Engineering and Knowlwdge Engineering, Capri, June 1992. 8. A. Bucci, P. Ciancarini, C. Montangero, “A Distributed Logic Language Based on Multiple Tuple Spaces”, Proceedings of the Logic Programming Conference, Tokyo, July 1991. 9. D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, C. Zaniolo, “The LDL System Prototype”, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 1, March 1990. 10. G. Clemm, L. Osterweil, “A Mechanism for Environment Integration”, *ACM Transactions on Programming Languages and Systems*, Vol. 12, No. 1, 1990.

11. R. Conradi, Ola Heensåsen, Svein-Olaf Hvasshovd, “EPOS DB Data model”, EPOS Project Report, Division of Computer Science Norwegian Institute of Technology, Trondheim, June 1987. 12. B. Curtis, M.I. Kellner, J. Over, “Process Modeling” *Communications of ACM*, Vol. 35, No. 9, September 1992. 13. M.Dowson, B. Nejmeh, W. Riddle, “Fundamental Software Process Concepts”, Proceedings of the First European Workshop on Software Process Modeling, Milano, May 1991. 14. M.A. Gisi, G.E. Kaiser, “Extending a Tool Integration Language”, Proceedings of the First International Conference on the Software Process, Redondo Beach, 1991.

15. C.A.R. Hoare, “Monitors: An Operating Systems Structuring Concept”, *Communications of ACM*, Vol. 17, No. 10, October 1974. 16. B. Holtkamp, H. Weber, “Object-Management Machines: Concept and Implementation”, *Journal of Systems Integration*, Vol. 1, No. 3/4, 1991. 17. G.E. Kaiser, N.S. Barghouti, “Database Support for Knowledge-Based Engineering Environments”, *IEEE Expert*, Summer 1988. 18. C. Liu, R. Conradi, “Process Modeling Paradigms: An Evaluation”, Proceedings of the First European Workshop on Software Process Modeling, Milano, May 1991. 19. A. Mahler, A. Lampen, “Integrating Configuration Management into a Generic Environment”, Proceedings of ACM SIGSOFT '90, 4th Symposium on Software Development Environments, Irvine, December 1990. 20. B. Peuschel, W. Schäfer, S. Wolf, “A Knowledge-Based Software Development Environment Supporting Cooperative Work”, *International Journal of Software Engineering and Knowledge Engineering*, Vol. 2, No. 1, 1992. 21. M.J. Rochkind, “The Source Code Control System”, *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4, December 1975. 22. W. Schäfer, S. Wolf, “Multi-User Support in the Process-Centered Software Engineering Environment Merlin”, Position Paper, Workshop on “Process Sensitive Software Development Environment Architectures”, Boulder, 1992. 23. A.

Sinha, “Client Server Computing”, *Communications of ACM*, Vol. 35, No. 7, July 1992. 24. R.N. Taylor et al., “Next Generation Software Environments: Principles, Problems and Research Directions”, COINS Technical Report 87-63, University of Massachusetts at Amherst, July 1987.
25. W.F. Tichy, “RCS - A System for Version Control”, *Software–Practice & Experience*, Vol. 15, No. 7, July 1985.