13

# A Reusable Software Artifact Library system as the core of a reuse-oriented software enterprise

*G. Jacucci*, E. Mambella†, G. Succi*, C. Uhrik‡, M. Ronchetti*, A. Lo Surdo†, S. Doublait†, A. Valerio**

* *Laboratorio di Ingegneria Informatica, DISA, Università di Trento, via F. Zeni 8, I-38068 Rovereto, Italia, tel. +39-464-443140, fax +39-464-443141*

† *Sodalia S.p.A., via Brennero 364, I-38100 Trento, Italia tel. +39-461-316111, fax +39-461-316663*

‡ *Faculty of the Dept. of Technology Programs, University of Phoenix 7800 E. Dorado Pl. , Englewood, CO*

*E-mail: gianni@lii.unitn.it*

**Abstract**

Introducing software reuse at a corporate level represents one of the most promising means of addressing the rising costs that are plaguing the software industry. A series of mechanisms are needed for shortening development cycles and providing reliable software of high quality which will be more maintainable and flexible for future extensions. This paper describes the experiences of Sodalia S.p.A., a young Italian company, in implementing such reuse methodology, particularly centred around a reuse tool specifically developed.

Since 1993, Sodalia's software engineers have been implementing a reuse program whose goal is making software reuse a significant and systematic part of the software process. The Sodalia Corporate Reuse Program is intended to institutionalize a software reuse process that incorporates reuse-specific activities all along the Sodalia object oriented software development process, drawing heavily on a reusable software artifact library system which has been designed to support the classification, management and search for artifacts to be employed in reuse efforts.

This paper presents an overview of the corporate reuse program implemented at Sodalia, focusing on the reusable software artifact library system and its role inside the reuse program.

A. Storr et al. (eds.), *Software Engineering for Manufacturing Systems*

**Keywords**
Software reuse, reusable software artifact library, reuse support organization

# 1 INTRODUCTION

Sodalia S.p.A. is a young Italian company that arose from a joint-venture between the STET Group (Italy) and Bell Atlantic Corporation (USA). The objective of Sodalia is the development of innovative telecommunications software products for the management and maintenance of telecommunication networks. The company's goal is to develop high quality, low-cost software and, more generally, methodologies which are able to deal with the even more complex and broad needs of the telecommunication sector, exploiting innovative development methods and technologies.

Due to the increasing competition and rapid technological innovations that in the past few years mark the field of telecommunications, this sector has undergone radical changes, forcing operators to cope with a growing demand of new applications and services, in terms of quality, variety, reliability, and, last but not least, low prices.

In this situation the traditional custom software development methodologies show all its inadequateness and ineffectiveness: to become really competitive in the telecommunication sector and able to survive in a global market, an organization needs to adopt leading edge technologies inside a defined, well planned and specific software development process. Moreover, traditional software development methodologies do not allow an efficient maintenance of the products, while any modification or extension of these products often results in a full re-development of them.

A new iterative development philosophy based on software reuse and using the emerging object oriented technologies seems to be the right trade-off among the opposing factors characterizing a successful telecommunication product or service: high-quality, low-cost, reduced time-to-market, flexibility and maintainability, just to name some. Software reuse is particularly interesting for Sodalia, since it develops similar applications for its parent companies and a new software system can be built reusing previously developed components.

Since 1993, Sodalia's Software Engineers have been studying a systematic reuse program to support and to integrate the software development activities, incorporating a reuse library to support the classification and management of the heterogeneous reusable components.

The next section of this paper deals with Sodalia's Corporate Reuse Program, while section 3 describes in detail the Reusable Software Artifact Library, highlighting its architecture and main functionality. Section 4 presents some experiences gained in the use of this reuse methodology inside Sodalia. In the last section some conclusions are outlined and future work is sketched.

# 2 SODALIA'S CORPORATE REUSE PROGRAM

The Sodalia reuse program aims at making software reuse a significant and systematic part of its iterative software development process. Due to the critical role played by reuse in achieving Sodalia's business objectives, and the novelty of the reuse technology, the reuse program will

be monitored and updated as the reuse experience grows, iteration after iteration.

The reuse program aims to develop a Software Reuse Process that incorporates reuse-specific activities within the Object-Oriented Software Development Process, and a reuse library to support the classification and management of reusable components. The strategy adopted to support systematic reuse within and across iterations and projects, organizes activities around the following two views: **Software Process for Reuse** and **Software Process with Reuse**, supported by a **Reusable Software Artifact Library** (RSAL).

A Metrics Definition Program has been set up with the goal of defining metrics for both software and reuse processes (SSPG 1994b). The first objective of the program is the definition of size and complexity metrics for measuring source code quality. Then, the metrics model will be extended to cover the analysis and design phases. Finally, a predictive metrics model will be conceived. The following sections expose the details of the corporate reuse program.

### *The concept of reuse for Sodalia*

Beyond the apparent obviousness of the concepts and objectives of reuse, several competing reuse approaches differ in at least one aspect: the artifacts they intend to reuse (e.g., the reuse of software, or of all life-cycle work-products). The following definition has been adopted to capture the essence of the Sodalia reuse approach:

> Software Reuse is the set of planned and systematic activities aimed at maximizing the use of existing software artifacts and known processes in the production and maintenance of new software artifacts.

Thus, Sodalia's reuse activity considers every kind of artifact, although more importance is given to high-level artifacts of the software life-cycle (e.g., requirements), since they hold the maximum information (see (Kain 1994), (Capers 1994), (Prieto-Diaz 1993)).

### *A Reuse-centered Software Development Process*

Sodalia's Software Engineering departments have defined and applied a reuse-centered software development process that fully supports software reuse and exploits the benefits of object-oriented analysis, design, and programming.

SIMEP (Sodalia's Integrated Management and Engineering Process (SSPG 1994a)) provides an advanced process architecture, modelling a highly iterative process as well as rapid prototyping. The iterative process model provides a rigorous framework for progression in the project by re-iterating a sequence of basic process steps. Moreover, the process architecture identifies reuse-specific activities (e.g., Problem Domain Analysis, Generalization of Requirements, etc.) as well as reuse-affected activities (e.g., coding) to foster reuse at all stages of the production process. The reuse-specific activities are described in the Sodalia's Software Reuse Guidelines (SSRG), (SSPG 1994c).

The SSRG "supports" SIMEP (reuse in-the-process) to provide guidance for proper conduct of a set of planned and systematic activities carefully defined and positioned along the development process to:

- **Develop with reuse**: maximize the reuse of already existing software components and artifacts;
- **Develop for reuse**: produce software with the highest reuse potential.

The SSRG "complements" SIMEP (reuse in-the-factory) to provide guidance for all those

activities, across and beyond projects, required to establish, maintain, and populate the RSAL, by producing specific components out of the projects. Moreover, the purpose of the guidelines for the activities of reuse in-the-factory, is to maximize the matching of reuse opportunity with reuse potential, through specific rules and recommendations to populate, search, extract, and maintain the reuse library.

### *Software process for reuse*

Software process for reuse includes the set of activities that allow, during the course of a project, the early identification of artifacts to be developed that might exhibit a high reuse potential either within or in other iterations of projects. Besides, added costs and time (incurred for reuse addressed specific activities) estimation techniques will be defined.

Sodalia pursues both **vertical reuse** (i.e., the reuse of software artifacts within a specific domain or application area) and **horizontal reuse** (i.e., the reuse of software artifacts across domains or application area). However, it puts major emphasis on the latter, since horizontal reuse provides the highest payoff.

Architecturally, Sodalia's products are viewed as a hierarchy of a number of product layers, each playing a well defined technical and market role. This architecture is defined by the Strategic Product Architecture (SPA) (SSPG 1993), whose goal is to set the strategic direction for structuring Sodalia's software products to support reuse activities. The SPA is organized in four layers:

- **Operating Environment** (layer 1): it addresses the technological foundation of the software products. The architectural components of this layer reflect the selection of basic database, graphics, and communications enabling technologies for the Operating Environment.
- **Application Shell** (layer 2): it characterizes software components specifically designed to solve application problems for a broad telecommunication market (e.g., Billing, Service Provisioning, Customer Network Management, etc.). This layer is likely to contain a large number of reusable software components.
- **Market Segment Specific** (layer 3): it reflects the specificity of a market segment related to a specific problem-domain (e.g., the Service Provisioning Application can be customized for small PTTs, large end-users, and mobile telephony market segments).
- **Customer Specific** (layer 4): it represents those software components which are highly specialized to address application features specific to a particular customer. Components at this layer have typically a low reuse potential.

The main architectural principle employed in SPA is relentlessly separating generic functions (i.e., that might exhibit a high reuse potential) from those which are specific to problem domains, market segments, and individual customers. The higher in the SPA a component is categorized, the less likely that component is to be reused. This concept is particularly true for horizontal reuse. However, within the same application domain (vertical reuse), components at all layers are equally highly reusable.

In application of the SPA, requirements for the components to be developed (at any level in the SPA) are examined to determine their reuse potential. The examination may recommend a "generalization" of a requirement to increase reuse potential. Since generalization should comply with market needs, it should be driven by stability and/or evolution analysis of the user requirements related to the market segment involved, as determined by the related business

plan. This critical activity is driven by the Problem Domain Analysis. The lower the level of the component, the more general is the analysis. On the contrary, the higher the level of the component, the more Customer-Specific or Market Segment-oriented is the analysis.

### *Software process with reuse*

Software process with reuse includes the set of activities that aims at maximizing reuse of existing components classified into the RSAL.

The approach supports the earliest possible identification (at requirements or design stages) of candidate reusable components since reuse is most effective when applied at early stages of the life-cycle. The principle, well practiced by hardware engineers, is to specify and design aiming at reusing existing components. The earlier these components are identified, the greater is the possibility to tailor requirements and design to reuse those components, rather than developing new ones. Since the definition of reuse is based on applying existing solutions to new problems, one can succeed in identifying something to reuse only if:

- The domain of available solutions is complete with respect to new, emerging problems;
- The description of both the problem and the solution are expressed at similar levels of granularity, for assessing their match.

If either one of these basic conditions is violated, it is practically certain that a new solution will be developed even for an old problem. Thus, the issue of anticipating the identification of reusable solutions depends on how early in the life-cycle one can satisfy the above necessary conditions. As a result, the reusable asset library can only be populated by artifacts certified according to the above conditions.

The identification of candidate reusable components must take place at each iteration (SSPG 1994a), moving top-down from large-grain artifacts (e.g., sub-systems) to fine-grain artifacts (e.g., classes), thus supporting the "reuse in the large" and "reuse in the small" paradigms.

Reuse "as-is" (i.e., without any alteration) and reuse "with change" (i.e., requesting prior modification/generalization/composition) have also been considered. Our strategy focuses on maximizing reuse as-is rather than reuse with change, because it maximizes productivity, relies on a certified quality, and minimizes (by factorizing) maintenance costs. Nevertheless, reuse with change should be considered when it is the only viable alternative. The latter implies the reprocessing of intermediate life-cycle artifacts of the component (specifications, design, test, etc.) before being able to reuse it. The cost of developing reusable components can be amortized only if the components are used repetitively with no or minimal changes.

## 3 REUSABLE SOFTWARE ARTIFACT LIBRARY

RSAL is a system for organizing rationally **artifacts** of a heterogeneous environment targeted to software production. The RSAL system supports the classification and search of various artifacts associated with a family of software development projects. By artifact, we mean anything with a self standing identity that can be found in the software life cycle: reports, user expectations, requirements, cost models, algorithms, programs are non exhaustive examples of artifacts. An artifact can be defined as the final result of an activity, including documents (e.g., High Level Architecture, User Manual) as well as software products (e.g., sub-systems,

classes).

A heterogeneous environment targeted to software production is by its own nature distributed in time and space, multi-systems and heterogeneous in terms of the machines and the operating systems working on it.

RSAL does not manipulate artifacts, but rather information on artifacts which are stored in software repositories, either local (i.e., owned by Sodalia) or remote (i.e., owned by Sodalia Parent Companies or others). Each artifact is described in RSAL by means of an artifact descriptor, whose two-fold purpose is to uniquely identify (locate) the artifact and to maintain a public description.

Artifacts have intrinsic attributes which pertain to their ontology, since they define their essence, e.g. the last opening date, the owner, the name, the i-node number are all intrinsic attributes of a Unix file. Artifacts have external attributes as well, which are annotations referring to artifacts and used in classifying and retrieving them: the quality level of a product, a description of the behaviour of a piece of code, a local comment on a remote file are all examples of annotations. RSAL handles both intrinsic attributes of its artifacts and external attributes. Moreover it allows to add further annotations to an artifact so as to make classification and search easier.

In RSAL there is a set of predefined attributes, i.e. attributes that are defined for any artifact. The set of such attributes includes the author, the creation date, the size and so on. Predefined attributes can be both intrinsic attributes, such as the creation date, and external attributes, such as the author: in such case whenever a reference to an artifact is inserted in RSAL such attributes must be filled and it is up to RSAL to keep them properly updated.

Attributes that are not predefined are called added.

Usual artifact descriptors such as faces, keywords, free text descriptions are regarded as attributes of an artifact.

Artifacts are connected by relations. A relation can be any kind of association that can be drawn between two artifacts such as one being the requirement of the other or one being the next version of the other and so on. Clearly, the definition of what is an artifact and what is a relation is not sharp: relations can rise to the level of artifacts when they have a self standing relevance, e.g. a baseline is a relation among several modules of code identifying a working product, therefore it is also a well defined artifact. Simplifying, RSAL may be thought as the catalogue of a library (a collection of cards describing in detail each book available in the library), and the artifact repository as the library itself. Before looking for a book in the library shelves, it is advisable to consult the catalogue. This is the principle adopted by RSAL. Each descriptor gives an exhaustive set of information for enabling the user in deciding if the artifact exhibits the characteristics he is looking for. The retrieval sub-system provides the user with powerful facilities for browsing the descriptor repository. Once the user has located the desired artifact descriptor, access to the artifact itself is provided by means of the tool used to create it or an equivalent one (supported tools).

## 3.1 RSAL architecture

RSAL offers an environment in which several users, working at their own workstations in different contexts, share the same pool of information. RSAL users see it as a central library whose retrieval facilities provide support in finding matches for all kinds of reusable artifacts

stored in the Sodalia's software repository.

RSAL organizes artifacts of several kinds under a uniform view. To do that, it manages references or pointers to artifacts, rather than the artifacts themselves. Attributes and relations are the primary parts of an artifact reference. RSAL stores the references to the artifacts in a metadata repository called the descriptors repository. This repository is directly accessed by the main modules of the system in order to archive, search, remove, change or retrieve an artifact descriptor. A Classification module is in charge of organizing the information about artifacts in a way that allows an efficient retrieval. An Identification module acts as a search engine on the descriptors repository, identifying the artifacts which satisfy given properties. This module is coupled and integrated with an interpreter handling SSQL queries -the SSQL module. RSAL produces some statistics on accesses to artifacts, and embeds an Announcement mechanism to inform end-users about new artifact acquisitions, new artifact versions, discovered bugs, etc. RSAL also performs application management functions to add users, set user privileges by category, and configure system parameters (e.g., keywords list). These last two functionalities are supported by the Application module.

A Tool Integration Platform underlying the whole system allows the interfacing with end-users and with application administrators, together with external tools, such as document tools or object oriented analysis and design tools.

Putting together these considerations, the RSAL high level architecture can be displayed as shown in figure 1.

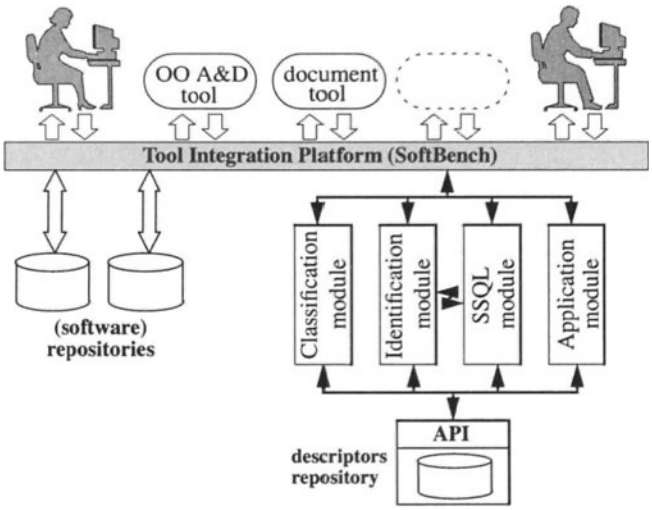


Figure 1: The RSAL general architecture.

## 3.2 Classification and retrieval

Classification of an artifact when it is inserted in RSAL allows the organization of all the attributes of the artifacts controlled by RSAL so that they can be efficiently and effectively identified and retrieved by the system. Effective reuse of existing artifacts requires a sophisticated classification/retrieval method. In the early stages of the reuse program, simple classification/retrieval methods are appropriate. However, more sophisticated ones are needed as the reuse library grows. Moreover, no single classification method is sufficient to find all relevant components of a given search (Frakes 1993b). Therefore, we consider multiple classification schemes, as explained below.

### *Classification of an artifact*

The three classification methodologies implemented in RSAL are different in the way the artifacts are classified and retrieved. When a new artifact is inserted, it has to be classified in all of the methodologies supported by the system. In fact, if this is not done, the artifact will not to be found when a search based on the missing methodology is performed.

For the **free text classification**, the artifacts are classified associating them with a text-description of arbitrary length which describes the artifacts themselves. For instance, a paper can be classified using its abstract.

In the **keywords classification** each artifact in the system is associated with a set of keywords that characterize the artifact itself and are chosen from a well defined dictionary. The dictionary can be expanded by the system administrator, who has the capability to remove unused keywords. For instance a class that implements a stack of integers could be classified using the keywords: integer, LIFO, source, C++, Object Oriented.

The classification of the artifacts in the **faceted** methodology is very similar to the keywords classification (Prieto 1989), (Sorumgard 1992). In fact, a set of keywords has to be associated each artifact in this case. The difference from the keywords classification is the dictionary from which the keywords are chosen: the dictionary is not plain, but structured into multiple facets that contain distinct subsets of keywords regarding different views of the artifacts. The single facets are structured as well. The keywords are stored in a weighted tree which represents their dependencies in terms of abstraction and specificity. The only user who can modify the facets is the system administrator. He can add or remove keywords, or move them to new locations in the trees; the system administrator can modify the weights in the trees, too, or even add or remove some facet. For instance if there are the facets Abstraction, Operates on and Dependencies, a stack of integers can be classified with the keyword Stack for the facet Abstraction, Integer for Operates on, and C++ for Dependencies.

### *Retrieval of an artifact*

An artifact stored in RSAL can be retrieved by navigating through the relations established between the artifacts or performing a search based upon one of the classification methodologies implemented in the system. The search is fundamental even when navigating through the relations - in fact, it is necessary to find an initial artifact from which to start the navigation. The search is done in different ways depending on the classification it uses: the user has to do a query in a format which depends on the different kinds of classification. If the results of the search are not satisfactory, the query can be relaxed or modified in some way and the search

will be repeated with the new query. For the free text classification, the query consists of a string that describes the characteristics of the artifacts sought, and the system returns all the artifacts in which the textual description contains the search substring. The only way to relax the query is to change the string to look for. To use the keywords classification, the query is prformed by giving the system a set of implicitly ANDed. The system will return all the artifacts with all the keywords of the query present in their set of keywords. The query can be relaxed by specifying less keywords or by changing them.

Using the faceted classification, the user has to specify an arbitrary number of keywords to look for in each of the facets, and the artifacts at the minimum distance (calculated from the structure of the facets) from the query are returned. The query can be relaxed by changing the keywords to look for, or adding new ones. In fact, if multiple keywords are specified for a single facet, this is intended as an OR between them and the minimum distance between them is considered.

## 3.3 The RSAL query language

RSAL allows a user access to a metadata repository linked to one or more artifact repositories via predefined menus and windows. This kind of access to the data is very powerful but suffers a limitation: it is frozen and can not be changed to accomplish all the desired functions a user can imagine. One would prefer to have a system which is fully programmable by the user so that it can be configured to satisfy even very unusual problems. This can be performed using a special language which permits interacting directly with data in the repository to perform efficient searches.

RSAL introduces a flexible query language called **Set-based Query Language** (SSQL). The aim of this language is to allow a user to program his own functions in order to interrogate the repository. In particular, a repository is naturally viewed as a collection of (generally) homogenous data. A language especially efficient for treating sets (Jayaraman 1987) is thus a natural choice to begin constructing a specific query language. We based SSQL on a set-based language called SL, which is derived from the Subset Equational Language (Jayaraman 1992), (Jayaraman 1988). SL is very efficient in dealing with sets and it is quite simple to use. In order to interface this language with the specific kind of data used by RSAL, we have expanded the basic instructions of SL so as to allow direct query over metadata in the repository.

The extensions made to SL concern the management of the repository, introducing the notion of metadata and its fields. It is possible to refer to a particular metadata field in order to analyse all the object descriptions for a particular value. This has been achieved by introducing two classes of operators: a set of specific operators and a general wide-range operator.

The specific operators deal with predefined attributes, both intrinsic and external, which describe every artifact in the system. These specific operators follow the syntax:

Value = predefinedAttributeName(EntityName)

where Value is the value of the predefined attribute predefinedAttributeName for the artifact EntityName. For each attribute present in the artifact descriptor, a specific operator is provided to test the indicated metadata field. As an example for the private data of the artifact, we have the operators: author, creationDate, type, location.

Also, a general wide-range operator is provided. The syntax of this operator is as follows:

Value = test(AttributeName,EntityName)

with the usual meaning for Value, EntityName, making it possible to refer to the given AttributeName in order to test its value.

Note that the possible inconsistency arising from, for example, referring to an attribute not defined for the artifact indicated, is properly handled by RSAL. It returns an error code if the attribute has been never defined in the metadata repository or if the bottom value of the domain of all possible values if the considered artifact has not defined a value for it.

Implementing a search with a SSQL program is very straight-forward. For instance in order to retrieve all the artifact created by the user John before 31 May 1994, the program clause should be:

searchedEntities(Author) = {X : X in RSAL; author(X) == Author,
early(creationDate(X), (31,may,1994))}.

and the query should be:

searchedEntities('John').

Ending this paragraph, the following clause retrieves the author, the type and how many copies are present in the repository of all the entities that have the title specified in the head of the query:

searchTitle(Title) = {[Author, Type, Copies] : Artifact in RSAL;
title(Artifact) == Title; Author is author(Artifact),
Type is type(Artifact), Copies is test(Artifact, 'copies')}.

# 4 REUSE EXPERIENCES

Designing and constructing reusable components requires additional effort estimated from 5 to 10 times the effort required to build non-reusable software components (Meyer 1994). There is a definitive need for an organization that supports and fosters reuse activities, since project teams cannot be relied on to achieve significant reuse: developers are necessarily focused on their particular applications and are constrained in terms of time, costs, and resources. Thus, project teams will not want to spend additional time developing software that does not directly affect their own application (Caldiera 1991).

## 4.1 Reuse Support Organization

A Reuse Support Organization (RSO) is a group of software engineering experts (senior analysts, designers, and programmers with consolidated experience in object-oriented technology) that provide support to project teams in the following way:

- Training and education: RSO defines and enacts the policies to disseminate the reuse culture and to encourage the adoption of reuse practices among project teams. It is principally through this mechanism that the organization learns how to "reuse" effectively.
- Provide expertise: whenever possible, RSO engineers work with the project teams to better support them in achieving both development "for reuse" and "with reuse".
- Development of reusable components: dedicated teams are in charge of producing reusable components by either re-engineering the "normal" software components or by developing new components whose reuse potential has been assessed during domain analysis. The act of

developing reusable components asynchronously with project team activities is called an "a-posteriori (or backroom)" approach.

- Maintenance of reusable components: reusable components may change as flaws are corrected and enhancements are made.
- Maintenance and administration of RSAL: the central repository is managed by a single team which controls the nature and quality of the information stored in the repository. This activity also deals with the population of the library of artifact descriptors, including setting policies for artifact acquisition and maintenance (e.g., evaluation, certification, classification, and weeding). In addition it has to define and maintain a coherent classification scheme for the stored artifacts, weeding old artifacts, announcing incoming ones, user management (e.g., add/delete users, set user privileges), and reporting on artifact usage.

Nowadays, the RSO provides both expertise and support to the project team activities, and development and management of the reusable software components. Currently, development for reuse activity performed by project teams is limited to the discovering for opportunities of reusable components. This information is given to the RSO that adapts, by generalizing or reengineering, the normal (not the reusable) artifacts. However, project teams perform development with reuse activities looking for opportunities of reuse among already developed and classified components. These activities are based on the RSAL tool and are supported by the RSO.

## 5 CONCLUSIONS

Many organizations, commercial, academic, and governmental, are devoting resources to software reuse, although software reuse is not yet a major force in most corporate software development programs (Frakes 1993a). Successful reuse introduction into industrial organizations has demonstrated that reuse benefits, such as improved productivity and reduced time-to-market, are really accessible and lead directly to lower cost, higher quality software.

The Reusable Software Artifacts Library RSAL has shown the great importance of a reusable repository approach to leverage the efficiency and payoff of a corporate reuse program, especially in an environment where heterogeneous, complex and strictly correlated artifacts are potential candidates for reuse. RSAL plays a critical role in the introduction of a formal and systematic reuse program within Sodalia, allowing reuse and support information to be shared inside the organization. Its scope embraces both vertical and horizontal reuse.

Presently, it is relatively easy to expand the built-in functionalities of the system (e.g., to expand recognized artifact types and their associated tools), but this in itself could be made more automatic in the future, thereby quickly augmenting the rate of expansion of the system and the number of interested users. In addition, the system maintenance features should evolve. For instance, there should be a natural way of testing consistency if changes have been made to a remote system, and a way of notifying a remote administrator when such an inconsistency is detected. As well, many other event triggered notifications and actions are easily imagined. Finally, some feature for learning and automatic performance improvement are envisioned, perhaps being able to better guess what ultimate artifact a user is seeking based on his or her present search or browse actions and records of the outcomes of all similar such search or browse actions from past users.

A large-scale experimentation of RSAL has been set-up in the form of a project sponsored by the European Community under the ESPRIT program (project TARSAL #20555). This trial application aims to verify the use of RSAL in three different software companies with different reuse maturity, different software engineering culture, different application domains, different operating systems and different size.

## 6 REFERENCES

Caldiera, G. Basili, V. 1991 Identifying and Qualifying Reusable Software Components, IEEE Computer, February 1991.

Capers, J. (1994) Economics of Software Reuse, Computer, Vol. 27, #7.

Frakes, W. B. (1993a) Software Reuse Survey Report, Software Engineering Guild.

Frakes, W. B. (1993b) RSL System Concept Definition, Software Engineering Guild.

Jayaraman, B. and Plaisted, D. A. (1987) Functional Programming with Sets, Third International Conference on Functional Programming Languages and Computer Architecture, Portland, 194-210.

Jayaraman, B. (1988) Subset-logic Programming: Application and Implementation, 5th International Logic Programming Conference, Seattle, 843-858.

Jayaraman, B. (1992) Implementation of Subset Equational Erograms, Journal of Logic Programming, volume 13, number 3, 299-324.

Kain, J. Bradford (1994) Pragmatics of reuse in the enterprise, Object Magazine, 3(6), pp. 55-58.

Meyer, B. (1994) Library Design, Tutorial Notes, Tools Europe 1994, Versailles (France).

Prieto-Diaz, R. (1989) Classification of Reusable Modules, Software Reusability, Volume 1, Concepts and Models, T. J. Biggerstaff and A. J. Perlis publisher, ACM Press

Prieto-Diaz, R. (1993) Status Report: Software Reusability, IEEE Software, Vol. 11, #3.

Sorumgard, L. S. and Tryggeseth, E. (1992) Classification, Search and Retrieval of Reusable Software Components, Division of Computer Systems and Telematics, Norwegian Institute of Technology.

SSPG (1993) Sodalia Software Process Group, Strategic Product Architecture (SPA), Internal Document, Sodalia Software Development Infrastructure.

SSPG (1994a) Sodalia Software Process Group, SIMEP: Sodalia's Integrated Management and Engineering Process, Rel.3, Ver.1, Sodalia Software Development Infrastructure.

SSPG (1994b) Sodalia Software Process Group, SIMEP Software Metrics Model, Rel.1, Ver.1, Sodalia Software Development Infrastructure.

SSPG (1994c) Sodalia Software Process Group, Sodalia's Software Reuse Guidelines, Rel.3, Ver.1, Sodalia Software Development Infrastructure.

## 7 BIOGRAPHY

**Gianni Jacucci**
He was born in Rome in 1943.
He received the Laurea in Physics (cum laude) from the University of Rome in 1967.
He had a research fellowship of the Ministery of Education (1968-1971) and was a researcher of the Italian C.N.R. from 1971 to 1986.
He is a professor of computational physics at the Faculty of Engineering of the University of Trento Professor of computer science at the University of Trento, an adjunct professor of physics and supercomputing applications at the University of Illinois at Urbana-Champaign, a visiting professor at Computer Science Department, Cornell University, and at Computer Science Department and National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.
He has many scientific publications.

**Eliseo Mambella**
He was born in 1931.
He received the university degree in Sociology (cum laude) from the University of Rome.
He was employee of SIP (now Telecom Italia) from 1953 to 1991.
He is author of the Business Plan for the constitution of Telesoft and consultant for the managing of Telesoft from 1992 to 1993. He is a reviewer of the EUROWARE ESPRIT project.
He is the director of the Department of Research and Technologies of Sodalia S.p.A. since 1993.
His working interests focus on software engineering and include software reuse, reengineering, software process.

**Giancarlo Succi**
He was born in Vercelli (I) in 1964.
He received the Laurea in Ingegneria Elettronica (cum Laude) in 1988 and the Ph.D. in 1993, both from the University of Genova, a M.S. in Computer science from State University of New york at Buffalo in 1991.
He is a researcher at the University of Trento since 1993.
He organized two post-conference workshops at the "International Conference on Logic Programming 1993" in Budapest, Hungary and at the "Joint International Conference and Symposium on Logic Programming 1992" in Washington, DC.
He is author of several scientific publications.
His principal interests are software engineering and declarative languages.

**Carl Uhrik**
He was born in Cedar Rapids, Iowa (USA) in 1957.
He received a Bachelor of Sciences (Electronic) (B.Sc.) from University of Texas A&M, College station, USA in 1980, a Bachelor of Computer Sciences (B.Sc.) from University of Texas A&M, College station, USA in 1981, a Master of Sciences (M.Sc.) from University of Illinois, Champaign-Urbana, USA in 1986 and a Ph.D. in Computer Science from University of Illi-

nois, Champaign-Urbana, USA in 1991.

Since 1990 he is a visiting researcher at University of Trento, Italy.

He was a visiting researcher at Computer Science Department and National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, USA, at National Center for Supercomputing Applications, Cornell University, USA.

He is author of many scientific publications.

**Marco Ronchetti**

He was born in Bolzano (I) in 1955.

He received the Laurea in Physics from the University of Trento in 1979.

He attended a Post-Doc at IBM-T.J.Watson Research from 1980 to 1981.

He is a researcher at the university of Trento since 1983.

He is author of 27 scientific publications.

He organized three workshops on scientific data graphic post-processing.

His recent interests are: simulation techniques in physics with supercomputers and workstations, distributed databases, image processing.

**Angela Lo Surdo**

She was born in Italy in 1953.

She received the Laurea in Mathematics from Rome University.

She worked for Italsiel S.p.A. (1978-1983) in development and maintenance of a TP monitor of IBM CICS, for Selesta Sistemi (1983-1990) on definition and coordination of the development and maintenance activities.

She was responsible for Configuration Management and System Administration from 1990 to 1991 and project leader in TCL domain area for Telesoft S.p.A. from 1991 to 1993.

Since 1994, she is responsible in Sodalia S.p.A. for Software Development and Project Management in the Research and Technology Department.

**Stéphane Doublait**

He was born June 27, 1963.

He received a Master of Sciences (M.Sc.) from Laval University, Quebec, Canada in 1990, a Bachelor of Sciences (B.Sc.) from Laval University, Quebec, Canada in 1986.

He was Principal Software Engineer at Bell Atlantic from 1990 to 1993.

He is Engineering Manager at Sodalia S.p.A since 1993, responsible for implementing the corporate reuse strategy of Sodalia.

He is author of more than 20 scientific publications in the area of artificial intelligence and software reuse.

His working interests focus on software engineering (software reuse, re-engineering, software process) and artificial intelligence (knowledge modelling & acquisition, automatic planning).

**Andrea Valerio**

He was born in Trento (I) in 1970.

He received the Laurea in Ingegneria Elettronica in 1995.

He is currently Ph.D. student at the University of Genova.

He is author of scientific publications.

His principal areas of interest are software engineering, especially software reuse, reuse metrics (both quality and productivity metrics) and declarative languages.