SENSEI: Software Evolution Service Integration

Jan Jelschen
Carl von Ossietzky Universität Oldenburg, Germany
jelschen@se.uni-oldenburg.de

***Abstract*—Software evolution tools mostly implement a single technique to assist in achieving a specific objective. Overhauling, renovating, or migrating large and complex legacy software systems require the proper combination of several different techniques appropriate for each subtask. Since few tools are built for interoperability, the setup of a toolchain supporting a given software evolution process is an elaborate, time-consuming, error-prone, and redundant endeavor, which yields brittle and inflexible toolchains with little to no reusability.**

This paper presents SENSEI, an approach to enable the implementation of an integration framework for software evolution tools using component-based, service-oriented, and model-driven methods, to ease toolchain creation and enable agile execution of software evolution projects. It will be evaluated by implementing and using it to build the toolchains supporting two software evolution projects, and having practitioners assess its usefulness.

I. INTRODUCTION

Large software evolution, migration, or reengineering projects usually require a combination of different techniques to analyze, reverse engineer, transform, and visualize (legacy) software systems under evolution. As each project has different goals, toolchains supporting their processes need to be tailored individually to their specific requirements [1]. Many tools exist, yet mostly only implement a single technique, and are usually not designed for interoperability. The lack of interoperability of software evolution tools is a general challenge of the field, recognized as such, e.g. by Müller et al. [2], Sim [3], Jin and Cordy [4], and Mens et al. [5].

Therefore, for each project, a toolchain has to be built by selecting the techniques required, finding appropriate tools implementing them, and then integrating these tools. With little to no means of interoperability, this involves creating a lot of *glue code* and data transformations to "wire up" all tools in the desired ways, a tedious and error-prone task. It yields brittle and inflexible toolchains, as extending or changing the toolchain, or swapping one tool for an alternative implementation, will require to also write new glue code. Consequently, this code is also non-reusable, as it is usually hard-wired to specific interfaces of the tools glued together.

The thesis presented in this paper proposes the SENSEI-approach (*Software EvolutioN SErvices Integration*), aimed at improving software evolution tool interoperability, and largely automate toolchain integration. Based on the fact that a large body of software evolution tools exist, yet they lack sufficient interoperability means to be easily integrated into the required, tailor-made toolchain, the following two objectives are derived: 1) Enabling software evolution practitioners to easily build toolchains tailored to their project-specific needs, focusing on the techniques to be employed, and the processes to be supported, while being as implementation-agnostic as possible, and abstract from interoperability issues. 2) Enabling tool developers to easily build tools with standardized, interoperable interfaces, or extend existing tools, with as little limitations to implementation technology choices as possible.

The approach taken towards these objectives is based on viewing software evolution techniques as *services*, to abstract from interoperability issues.It entails 1) surveying tools and techniques, and compiling them into a **catalog of standardized software evolution services**, 2) utilizing existing, component-based technology to provide an **integration framework** using the catalog as a basis, and 3) providing a means to describe software evolution processes in terms of coordinated services, and to **automatically generate toolchains** based on the integration framework.

This paper is outlined as follows: Before describing SENSEI in detail, Section II presents related work, and highlights differences to similar approaches. Section III introduces the proposed solution to the software evolution tool interoperability problem. Current and ongoing work is presented in Section IV, followed by an evaluation method in Section V. Section VI sums up expected contributions of the thesis.

II. RELATED WORK

Sim [3] distinguishes between three levels of tool interoperability, with ad-hoc interaction on the lowest level, exchange file interaction in the middle, and (API-based) dynamic interaction on the highest, most desirable level, which SENSEI is aimed at achieving. Currently, there are standard exchange file formats such as GXL [6], and tool suites (e.g. Bauhaus [7], Moose [8]), which are extensible, but require tools to be built specifically for their environment.

A project sharing some ideas with SENSEI is *SOFAS* [9], also aimed at tool integration by orchestrating services found in a catalog. However, the service term of SOFAS is closely related to the chosen technology (RESTful web services), whereas SENSEI tries to provide a technology-independent concept, and a correspondingly generic service catalog. SOFAS is restricted to analysis activities, explicitly exploiting their uniformity and thereby excluding, for example, transformations (restructurings, refactorings). SENSEI aims at supporting all software evolution activities and involved techniques.

Within the *SENSORIA* [10] project, concepts for model-driven generation of integrated software systems from higher-level descriptions like BPEL-based orchestrations [11] have been developed. A similar approach is taken by SENSEI to automatically derive toolchains from process-oriented descriptions. SENSORIA was neither focused on software evolution, nor was tool interoperability its central concern, though. The work by Biehl et al. [12] is similar to SENSEI in several regards, e.g. it is based on services and their orchestration, and model-driven techniques to derive executable toolchains. They address the domain of embedded software development. Discovery and description of relevant services, their cataloging and their standardization is not covered.

A central part of SENSEI is the creation of a comprehensive, rigorous catalog of services relevant to software evolution. There are no surveys of techniques covering the entire field of software evolution, only specific sub-areas or groups of techniques, e.g. clone detection [13], refactoring [14], or slicing [15] are covered. Naturally, neither are such surveys tailored towards service identification, nor do publications on specific techniques describe them in terms of services. There are taxonomies of software evolution or sub-fields, e.g. Buckley et al. [16], which aid the organization of the field, but do not provide an in-depth survey of relevant techniques.

In summary, approaches for tool integration in domains other than software evolution exist, and are partly built around ideas which are also part of SENSEI. Using services and their orchestration as high-level descriptions, and model-driven techniques to derive toolchains has therefore been proven a viable for tool interoperability. However, these approaches have been tailored for different application domains, are less generic than SENSEI, or are limited to subfields of software evolution. Moreover, no service standardization is offered, limiting reusability and flexibility, e.g. to exchange one implementation for another without changing integration logic. To achieve this level of interoperability, SENSEI aims to establish a catalog of software evolution services, including a seamless approach to the orchestration of appropriate implementations.

III. APPROACH

To give an overview of SENSEI, the following is a description of the different artifacts of the approach, and how they relate to each other: First, there is the *service catalog*, containing rigorous descriptions of software evolution services. With this, the processes of a software evolution project in need of tool support can be specified by selecting required services from the catalog, and describing the way in which they should interoperate as *workflows* in an appropriate language. The *service-component registry* provides a mapping between abstract services and concrete implementations by tools. The *tools* are encapsulated in, or developed as, *components*, conforming to the component model of the underlying framework. A set of model *transformations* embodies the mapping to a specific component framework. With all these artifacts, a *model-driven code generator* produces the required *platform-specific glue code* which realizes tool integration. The final result is a set of tools integrated into an application framework to execute the specified software evolution processes.

A deliberate distinction is made between services and components (cmp. e.g. [17]). The service term is used to refer to abstract descriptions of functionalities. Components are viewed as concrete implementations of provided services. Moreover, the use of service-oriented principles should not be taken to imply *SOA* (cmp. [18]), *SaaS*, or *cloud computing*. Due to confidentiality issues, SENSEI is expected to be used to produce toolchains which can be fully controlled by the organizations owning the systems under evolution.

SENSEI is thus based on concepts from *service-oriented*, *component-based*, and *model-driven* software engineering: **Service-orientation** provides high-level, implementation-agnostic descriptions of software evolution techniques. **Component-based** frameworks offer the necessary uniformity to plug in service-providing tools in a generic way. **Model-driven** technology provides the link between the two, by generating concrete toolchains from services orchestrated to support desired processes using model transformations.

Using these three views, the approach and its challenges are explained in more detail in the following sections, with deliverables highlighted in italics.

A. The Service-Oriented View

Taking a service-oriented view towards software evolution activities lies at the very center of SENSEI. Because services reside on a conceptual level, they are technology-independent, which hides interoperability concerns. This only defers the integration question to a lower level, but it allows to define meaningful services without regard to their implementation, and focus on the task of designing processes supporting software evolution projects (Objective 1 in Section I). To enable this, a *service description meta-model* [19] has to be created, naming all relevant information necessary to choose and use a service, among them its semantics, input and output artifacts and their datatypes, as well as classifications into categories, to ease service discovery. Also, a means to associate services with *capabilities* is required, allowing to specify abstract services, e.g. having a single metrics calculating service in the catalog, instead of one service for each metric (see Sec. III-C).

Next, the description meta-model has to be populated with data, i.e. a survey of relevant software evolution tools and techniques has to be performed [20], to extract services and compile them into a *service catalog*, a central deliverable of the thesis, on which the remaining parts of SENSEI are based on. The model serves as a template to describe all services of the catalog consistently. While surveying, it would be natural to also perform a classification of identified services, yielding a *taxonomy* which aids the catalog's user in finding and choosing the right service for a given task. The service-oriented view is a prerequisite for leveraging workflow technology for service orchestration, based, e.g. on a language like BPEL [21].

B. The Component-Based View

The service-oriented view of SENSEI abstracts from concrete implementations. The component-based view complements this with a tangible technology basis for its realization. A *component registry*, described by the *component description meta-model*, is used to map services to providing components, with concrete capabilities specified (cmp. Sec. III-C).

SENSEI is designed to be generic with respect to the chosen implementation technology, however, the choice does have an impact on Objective 2 (Sec. I), i.e. how easy it is to integrate (existing) tools. A challenge is therefore the elicitation of *requirements* for a component model and framework, which can support software evolution service integration. To ease integration of existing tools, which are based on completely different technologies and platforms, a suitable integration framework should be platform independent, supportive of diverse integration and implementation technologies, and allow distribution over a network. To execute processes defined in a language like BPEL, it should also allow to incorporate workflow engine technology. After comprehensive evaluation [22], [23], the most promising candidate to serve as target infrastructure is SCA [24], a set of standards for building service-oriented and component-based applications. Based on SCA, a working *prototype* will be implemented.

C. The Model-Driven View

The model-driven view to SENSEI "ties it all together": Here, the high-level artifacts of the service-oriented view – services and process definitions – are taken and mapped to implementing components and an interoperability framework provided through the component-based view. In a way, the opposite is also true: the model-driven view *decouples* service-oriented view from the component-based view, thereby allowing to replace the technologies chosen for implementation, or parts thereof, while the conceptual layer remains untouched.

To leverage model-driven technology, all required information has to be available as models, conforming to appropriate meta-models. SENSEI defines four integrated meta-models, some of which have already been introduced:

1) The *service description meta-model*, as already introduced, is used to describe services and create a service catalog. 2) The *component description meta-model* is used to register tools implementing services, forming a component registry. 3) The *orchestration meta-model* (possibly instantiated by an existing process language) is needed to describe software evolution processes as workflows, i.e. how services should be coordinated to form the desired toolchain (Objective 1). 4) The *service capability meta-model* is used as part of all other models for different purposes. In the service catalog, it allows to specify abstract reference services with sets of possible capabilities (e.g. the actual metrics a metric calculating service can provide). In a workflow, required capabilities are listed for each referenced service, and in the registry, the capabilities actually provided by a component are defined.

Using model transformations, these high-level artifacts will automatically be turned into platform-specific integration code, by matching up selected services with implementing components, respecting specified capabilities, and deploying the workflow definition to a workflow engine.

IV. CURRENT AND ONGOING WORK

To become familiar with the different fields relevant to the thesis, extensive literature research into software evolution and its sub-fields, as well as into service-oriented, component-based, and model-driven software development has been performed. A first sketch of the approach was published in 2011 [25]. A number of available service-oriented or component-based frameworks have been reviewed [23], and some feasibility experiments have been performed [26], [22].

Current work is focused on framing and populating the service catalog, arguably the centerpiece of the thesis, from which the other parts derive. To create the catalog, relevant software evolution services have to be discovered, first, and then described with all necessary information to implement, find, and use them. This is done by an extensive literature survey, taking into account over 3,000 publications from the field of software evolution published in roughly the last 15 years, and using text-mining techniques like automatic clustering and classification to group and filter out those publications likely to describe services. A first, simple service description meta-model has been created to capture the information gathered this way. This approach, along with early results, has recently been presented at a workshop [20]. The service description meta-model has since evolved, and is the subject of a report presented at "CoNaIISI" [19].

The literature survey, and with it the first version of the service catalog, is expected to be concluded within the year 2013. Further work packages will be concerned with implementing the prototype, extending it with the model-driven superstructure in a separate step, evaluating SENSEI, and writing the actual thesis, for a submission in mid-2015.

V. EVALUATION

As a proof-of-concept, the SENSEI-approach will be prototypically implemented. To demonstrate its benefits, it is planned to be used in the context of the following two projects:

Q-MIG. This project [27] is aimed at building and evaluating a *quality-driven, generic toolchain for software migration*. An industry-provided set of tools for reverse engineering and COBOL-to-Java migration will be embedded in the SENSEI-framework, complemented by tools for measuring, monitoring, and comparing software quality metrics evaluated at reading points situated in between successive migration steps. The project aims at investigating the impact language migrations have on a software system's quality. The quality measurement can also be used to compare different migration strategies, or migration tools – provided they can be easily swapped against each other, a feature SENSEI is expected to provide.

Experts from industry will participate in this project, and will be asked to assess the expected benefits of using SENSEI, compared to setting up a project's tool support without it.

Energy-efficient applications. Software evolution techniques like dynamic and static analysis, and refactoring, can also be used to monitor and rate an application's energy consumption, detect *energy code smells*, and subsequently remove them. To support the study of different research questions in this area (cf. [28], [29], [30]), a suitable tool infrastructure has to be built, combining tools for static and dynamic code analysis, metrics evaluation, refactoring, and visualization, as well as "helper" tools, e.g. for parsing, unparsing, or data transformation. This research focus therefore provides the second application for SENSEI's evaluation.

VI. Expected Contributions

This paper proposed a PhD thesis centered around developing an approach towards better tool interoperability in software evolution. SENSEI has been conceived around modern software engineering principles, making use of concepts borrowed from *service-oriented*, *component-based*, and *model-driven* software development. It is designed to make use of existing technology as much as possible. The main products to be created as part of SENSEI and the planned thesis in general are: 1) A *service description meta-model*, providing a taxonomy and a classification scheme of software evolution activities and techniques. 2) A software evolution *service catalog* instantiating this meta-model, filled by a comprehensive, structured literature review, naming, describing, and standardizing services used in the context of software evolution. 3) An *integration framework*, utilizing existing, component-based technology for the target platform, and a model-driven layer as both the link to, and the decoupling from, the conceptual service level.

With these contributions, the following are regarded as the central expected benefits: First, the provision of an integration framework for software evolution tools is expected to ease project execution in several ways. The ability to design project workflows solely based on services frees practitioners from having to worry about technical issues, allowing them to focus completely on their main tasks. The automation of tool integration frees up time and engineering capacity, to get more actual work done, or execute software evolution projects more cost-efficiently, or in a more timely manner. The flexibility gained through automatic integration also enables more agile processes, giving room for experimentation and the ability to properly react to unexpected obstacles.

The model-driven approach provides an abstraction layer between concept and technology level. This decoupling permits the realization of the SENSEI framework using different technological spaces, and facilitates independent evolution of both levels. Through the service catalog and taxonomy, a comprehensive, structured overview of the whole field of software evolution, and all major techniques is contributed to the (research) community. This will organize the field, make differences and similarities of techniques visible more plainly, and help identify synergetic potential and avoid redundant developments. In general, it gives a clearer picture of past and present research, and the state of the art. It can show research trends and reveal opportunities for further research.

SENSEI will be validated by applying it in the context of two software evolution projects, one including an industry partner, whose software evolution experts will help to assess the value of using the approach.

References

[1] J. Borchers, "Erfahrungen mit dem Einsatz einer Reengineering Factory in einem großen Umstellungsprojekt," *HMD Themenh. Migr.*, 34(194), pp. 77–94, Mar. 1997.

[2] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong, "Reverse engineering: a roadmap," in *Proc. Conf. Futur. Softw. Eng.*, New York, NY, USA. ACM, 2000, pp. 47–60.

[3] S. E. Sim, "Next generation data interchange: Tool-to-tool application program interfaces," in *WCRE*, 2000, pp. 278–280.

[4] D. Jin and J. R. Cordy, "Ontology-based software analysis and reengineering tool integration: the OASIS service-sharing methodology," *21st IEEE Int. Conf. Softw. Maint.*, pp. 613–616, 2005.

[5] T. Mens, M. Wermelinger, S. Demeyer, R. Hirschfeld, S. Ducasse, and M. Jazayeri, "Challenges in software evolution," *Proc. Int. Work. Princ. Softw. Evol. (IWPSE 2005)*, pp. 13–22, 2005.

[6] R. C. Holt, A. Schürr, S. E. Sim, and A. Winter, "GXL: A graph-based standard exchange format for reengineering," *Sci. Comput. Program.*, vol. 60, no. 2, pp. 149–170, 2006.

[7] A. Raza, G. Vogel, and E. Plödereder, "Bauhaus–a tool suite for program analysis and reverse engineering," in *Reliab. Softw. Technol. – Ada Eur. 2006*, LNCS, 4006. Heidelberg: Springer, 2006, pp. 71–82.

[8] S. Ducasse, T. Gîrba, and O. Nierstrasz, "Moose: an agile reengineering environment," *ACM SIGSOFT Soft. Eng. Notes*, 30(5), pp. 99–102, 2005.

[9] G. Ghezzi and H. C. Gall, "A framework for semi-automated software evolution analysis composition," *Autom. Softw. Eng.*, 20(3), pp. 463–496, 2013.

[10] M. Wirsing and M. Hölzl, Eds., *Rigorous Software Engineering for Service-Oriented Systems*, LNCS 6582. Springer, pp. 541-560, 2011.

[11] L. Gönczy, A. Hegedüs, and D. Varró, *Methodologies for model-driven development and deployment: an overview*, LNCS, M. Wirsing and M. Hölzl, Eds. Heidelberg: Springer, 2011, 6582.

[12] M. Biehl, J. El-Khoury, F. Loiret, and M. Törngren, "On the modeling and generation of service-oriented tool chains," *Software and Systems Modeling*, pp. 1–20, 2012.

[13] R. Koschke, "Survey of research on software clones," in *Duplic. Redundancy, Similarity Softw.*, R. Koschke, E. Merlo, and A. Walenstein, Eds., vol. 06301. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, 2006.

[14] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Trans. Softw. Eng.*, 30(2), pp. 126–139, 2004.

[15] F. Tip, "A survey of program slicing techniques," *J. Program. Lang.*, 3(3), pp. 121–189, 1995.

[16] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *J. Softw. Maint. Evol. Res. Pract.*, 17(5), pp. 309–332, 2005.

[17] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing Evolving Services," *IEEE Softw.*, v28(3), pp. 49–55, 2011.

[18] A. T. Manes, "SOA Is Dead; Long Live Services," 2009. [Online]. Available: http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html

[19] J. Jelschen, J. Meier, M.-C. Ostendorp, and A. Winter, "A Description Model for Software Evolution Services," in *CoNaIISI'2013*, ISSN 2346-9927, Cordoba, Argentina, Nov. 2013.

[20] J. Jelschen, "Discovery and Description of Software Evolution Services." *Softwaretechnik-Trends*, 33(2), pp. 59–60, 2013.

[21] OASIS, "Web Services Business Process Execution Language Version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[22] J. Meier, "Eine Fallstudie zur Interoperabilität von Software-Evolutions-Werkzeugen in SCA," Bachelor's thesis, University of Oldenburg, 2012.

[23] M. Ringe, "Vergleich komponentenbasierter Frameworks zur Werkzeug-integration," Master's thesis, University of Oldenburg, 2013.

[24] OASIS, "Service Component Architecture Assembly Technical Committee," 2013. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=sca-assembly

[25] J. Jelschen and A. Winter, "Towards a Catalogue of Software Evolution Services," *Softwaretechnik-Trends*, 31(2), 2011.

[26] J. Jelschen and A. Winter, "A Toolchain for Metrics-based Comparison of COBOL and Migrated Java Systems," *Softw. Trends*, 32(2), pp. 67–68, 2012.

[27] Software Engineering Group of Carl von Ossietzky University and pro et con Innovative Informatikanwendungen GmbH, "Q-MIG: Building a Quality-Driven, Generic Tool-Chain for Software Migration," 2013. [Online]. Available: http://se.uni-oldenburg.de/Q-MIG

[28] J. Jelschen, M. Gottschalk, M. Josefiok, C. Pitu, and A. Winter, "Towards Applying Reengineering Services to Energy-Efficient Applications," in *16th Eur. Conf. Softw. Maint. Reengineering*, 2012, pp. 353–358.

[29] M. Gottschalk, M. Josefiok, J. Jelschen, and A. Winter, "Removing Energy Code Smells with Reengineering Services," in *Informatik 2012*, U. Goltz, M. Magnor, H.-J. Appelrath, H. K. Matthies, W.-T. Balke, and L. Wolf, Eds., LNI 208. Bonner Köllen Verlag, 2012, pp. 441–455.

[30] M. Gottschalk, J. Jelschen, and A. Winter, "Energy-Efficient Code by Refactoring." *Softwaretechnik-Trends*, 33(2), 2013.