Abstract—Software reuse practices and tools have been proposed over the last three decades. From the reuser’s point of view, it is necessary to provide facilitators in order to execute reuse tasks among tools. We propose the use of Reuse Assistants (RA) as a representation for tasks that tailors software assets to be used in chained executions of reuse tools. Since many tools do not interchange their input and output (IO) with others, reuse processes are limited to executing reuse tasks from a single tool. Moreover, reusers are required to manually adapt IO parameters between tools aiming to execute a more complete reuse process. However, such adaptations are subjected to inconsistencies and errors. In this sense, by specifying RAs through a common representation one would be able to execute a reuse process by chaining tools in an assisted way. This paper presents a work in progress to support software asset tailoring through RA.

Keywords- Reuse technique; Software reuse; Reuse process; Tool chain; Reusable Asset Specification - RAS

I. INTRODUCTION

Software reuse has been used over the last three decades as a practice that leads software engineers to leverage on past experiences while creating new software systems [20]. As a result, several techniques and their corresponding reuse tools were developed, such as Product Line Architectures (PLA) [12][14], Component Based Development (CBD) [28], source-code generation based on Model Driven Development (MDD) [4][22], and Object-Oriented Framework Instantiation (OOFI) [23]. In this scenario, some tools are used together to develop a software project. However, most of the time they have incompatible input and output (IO) parameters (software assets), making their execution in a chain of reuse tasks complex. Thus, reusers are challenged to manually chain tools by adapting these assets in an error-prone task. Accordingly, this paper presents a reuse scenario composed by existing solutions (a toolset) that promotes software reuse with different reuse techniques. Therefore, we present a proposal to represent assistants that tailors software assets to be interoperated among reuse tools.

Software engineering practitioners commonly use more than one tool and technique to support activities that involve reuse. Such activities are executed using some software assets that are refined from higher abstraction levels to specific needs [3]. In this context, Aho et al. claim that, even in the case where a single technique is used to support such activities, interchanging information between reuse tools is a hard and error-prone task [1]. Authors exemplify a scenario where many tasks are required to support the development of web services functionalities based on MDD technique such as to design a UML model, to generate source-code for Java and SQL schemas, and to finally integrate everything and deliver a testable prototype. Each activity is supported by a specific tool and is manually synchronized by technical stakeholders such as developers and project leaders.

In this context, this work proposes a common representation to specify reuse tasks among tools as Reuse Assistants (RAs) [7]. RAs correspond to an EMF based model [9] that represents in high-level of abstraction the executable task required to adapt software assets to be used among tools. Examples of assets are programming APIs, model transformers or software domain designs. In this direction, we provided in [7] a common representation for RAs to allow tools chain as an extension of Reusable Asset Specification (RAS) [27]. RAS supports some software activities with descriptions and guidance tasks, helping reusers in performing reuse independently from a tool support.

In this sense, this paper presents a proposal to specify and adapt RAs to different execution environments. Experiences on practical application regarding these adaptations are discussed as well as ongoing work. Preliminary results are promising and suggest that the usage of reuse techniques among tools can be facilitated though the proposed approach.

Next sections are organized as follows. Section II provides an introduction about reuse assistants with motivation based in industry experiences. This section also presents an example of tools chain and points to the lack of current approaches to support reuse processes among tools. Section III presents a proposed architecture and reports experiences as means to validate the proposal. Section IV discusses the related work, and Section V presents some concluding remarks and limitations of our work.

II. MOTIVATION

We have large experience in supporting reuse techniques [4][12][28][13][33]. More specifically, we tailor tools and techniques to support model-driven based processes for different development scenarios. These processes require dealing with a set of reusable assets manipulated and executed with some tools. In this context, it is important that each activity be correctly executed and to ensure that assets used as input for further tasks are in conformance. This implies in a situation where tools must interoperate these assets with valid IO parameters. Besides, each activity requires variant guidelines to help stakeholders in executing transformations.
These are dependent on the adopted reuse techniques involved in each activity and the tools used towards modeling and source-code generation practices.

A. Illustrative Reuse Scenario

In this sense, the following activities can compose a reuse scenario that requires asset adaptations among tools:

1. Functional requirements (asset 0) are specified using a word processing template. Alternatively it could be specified with SPEMArti [29], a Domain Specific Language (DSL) to describe use cases.
2. A UML model (asset 1) is designed with Enterprise Architect (EA). Other alternatives to EA are also used and each one exports and imports models using different versions of XMI.
3. A second tool named MockupToME [6] is used to generate graphic user interfaces (asset 2) taking as input asset 1. Alternatively, asset 1 can be imported by applying a reverse engineering using WCT module for Java reverse engineering from code to model [32] or be generated by a PLA approach such as Odyssey [12] or RDL-FI [13]. Besides, in case of asset 0 being specified with the DSL then the generated user interfaces must be traced to asset 0.
4. A database schema is modeled using an Object-Relational Mapping (ORM) UML Profile [19] (asset 3). Alternatively, one can use WCT wizards to help in the application of some profiles. In any case, it is necessary to use guidelines because this task requires a high know-how to execute.
5. A model transformation (asset 4) is executed with FOMDA [4] to generate: a) SQL scripts (asset 5), used to create database schemas; b) entity classes (asset 6) annotated with ORM mapped for Hibernate [31]. Other transformation tools can be used as alternative.
6. SQL scripts are imported into an SGBD, which creates a database. Alternatively, one can leverage the database creation to Hibernate. In this case SQL scripts generation is optional as well as this step.
7. Other model transformers (assets N) are executed to generate each web application layer such as view, controller, validation and data access objects. Alternatively, some applications support Java Swing. In this case, the generation of graphical user interfaces for Java Swing and also a remote layer are required.
8. Finally, the generated source-code is changed by developers. Here, many assets are used such as Java APIs, existing source-code, documentation, etc.

B. Difficulties to Interoperate Reusable Assets Among Tools

Based on such activities, we can highlight some difficulties to chain assistants among tools and techniques: a) they support reuse activities that require different asset as inputs [15][11][30][25]. Some input must be changed after generated; b) tools are developed to interchange files and not to interoperate reuse tasks [24]. This was the reason why in [33] Wizards were developed to help stakeholder in a UML Profile application: UML modeling tools do not provide a simple solution for specifying it; c) incompatibility between input and output (IO) [1], which required manual adaptations from XMIIs generated by UML tools to be executed by model transformers; d) tools do not allow to modify existing reuse tasks because most of the time they are black-box [23]. In this case, wizards must be developed outside reuse tools.

Due to the lack of appropriate guidance, important reuse techniques may be missed. This situation occurred when a company withdrew our proposed MDD techniques and tools to program everything by hand [5][6]. This was motivated by the following reasons: 1) It was required a long period of time to a modeler to learn a UML tool, when executing the second task, and by dealing with valid XMI files as input for transformations in the fifth task. Here we learned a lesson, that even in equal versions of XMI, tools export models differently. As consequence, model transformers did not work properly; 2) The learning curve was even bigger to execute the fourth task, because the application of UML Profiles requires experienced modelers. Here we learned that facilities such as guidelines help inexperienced modelers. However, they are costly to develop as wizards; 3) We learned about how to execute further activities after executing each model transformation (fifth and seventh tasks), such as to correct source-code pieces. With no guidance to pinpoint what should be changed in each generated source-code for each application layer, developers were confused about the required procedures to apply glue-code.

To fix these problems, we provided wizards as a solution for guidance and execution support [6]. This facilitated the proposed activities adoption by non-experienced stakeholders. Unfortunately, this scenario is not well explored by software engineers, leveraging reuse assistants as options to support reuse activities. Accordingly, in absence of a common representation for reuse assistants, reuse tools are strictly adapted by people who have strong know-how about the internal structure of the software, because a wizard is typically a black-box program. In this sense, it is not possible to know the activities and the assets generated by each assistant. As consequence, inconsistencies are faced in daily practice regarding MDD.

C. Desirable Scenario

Currently, MDD practitioners must manually adapt reusable assets interoperated among tools in an error-prone process [1]. The reason is that a non-common representation for reuse activities exists to interoperate these IO parameters. They change IO file formats and apply corrections to fit each software asset with activity parameters. This requires abilities, usually driven to highly technical stakeholders (i.e., software developers). However, to change this scenario, we claim that reuse assistants specification, which tailors software assets, should be specified by non-technical stakeholders [23].

This would imply in a common representation for reuse tasks, where each reuse tool interoperates simple information such as: 1) IO parameter with some constraints; 2) descriptions and guidance of assets; 3) reuse activities execution support. Besides minimizing incompatibility errors, it would also bring other benefits, such as to automate software process. Moreover, these assistants could be chained based on CBD techniques and be interoperated using existing task execution
tools. Also, a common representation would allow a deployment of reuse processes into existing development environments such as Eclipse. However, tools are not prepared to support this kind of approach.

III. PROPOSED APPROACH AND EXPERIENCES

In direction to facilitate the execution of reuse tasks among tools, we propose the architecture shown in Figure 1 that support reuse with three main steps: 1) asset specification; 2) asset acquisition; 3) and asset usage and execution among tools. The first step is to specify assets in a common representation and store it into a Reusable Asset Specification (RAS) based repository, discussed in Section III A and B. In the second step, the reuser acquires these assets by searching the repository as depicted in Section III C. Finally, a program called RAS Client Deployer configures user workspaces using assistants as input for transformations, as detailed in Section III D.

![Figure 1: Proposed Architecture to Support Reuse Assistants](image)

Existing reuse tools must be plugged into workspaces without recurring to black-box configurations such as those required by IDE plug-ins [10][21]. For this reason, we have decided to define assistant deployment information as a RAS model [7], since it allows detailing how artifacts must be used in a reuse context. Accordingly, we propose that assistants must contain a configuration about reuse practices and about the deployment of their own artifacts. Besides, it also serves as a protocol between an asset repository and a user workspace. Therefore, any tool that supports our extended RAS metamodel can use a reuse assistant to configure a user workspace.

A. Reuse Assistant (RA) Model Specification

Figure 2 illustrates the activities used in the “Specification” step towards designing reusable software elements. Thus, it is important to know the differences between these elements. Accordingly, following definitions are important.

1) **Artifacts** are elements produced and consumed along software process lifecycle. Examples are application models, APIs and libraries, source-code, documentation, reuse tasks as tutorials and how-to, etc.

2) **Assets or Reusable Assets** are elements that document one or more artifacts. They provide guidelines that describe reuse activities required by artifacts to be adapted and used. These elements are described with standard RAS metaclasses and are packaged in a file composed by a single or a set of artifacts, as illustrated in Figure 1 by the “RAS Asset” package. However, despite RAS provides a rich set of data to detail reuse activities, currently this standard has no support to specify details such as execution and deploy support. Therefore, extensions for reuse assistants are required.

3) A **Reuse Assistant (RA)** is a model representation about execution and deploy of reuse activities that allows adapting one or more reusable assets to support interoperation among different reuse tools. This means that it allows: a) to organize a set of tasks that can be performed by (re)users, with the help of a provided information through RAS assets; b) and/or it can be executed with task and process execution languages available, for example, in Eclipse IDE workspaces; c) and/or it configures reuse tools to support the asset development and refinement. This means that those assets discussed in Section II A must have a common representation (as a RA) to be adapted and interopereated among those reuse tools.

4) **Reuse tools** are any executable program/application that aims to facilitate the reuse of software artifacts. These programs can be chained in software production tasks, which can be represented in higher abstraction levels as reuse assistants.

A reuse assistant is designed to facilitate reuse of software artifacts. It is used to describe one or more reusable assets. Information about RA must guarantee that reuse activities (manual or executable), and artifacts are installed in the right place, configuring the supporting tools/plug-ins contained in a workspace. Thus, we assume that this kind of task is executed in the (re)user local machine with a focus on defining a few artifact specializations, such as the examples depicted in [23], or generating a source-code fragment, as exemplified in [4].

![Figure 2. Activities in Support for Asset Specification](image)

B. RAS Based Repositories

Activities shown in Figure 2 are executed by an asset producer. Thus, first two are used to specify artifacts, assets and assistants in a RAS compliant metamodel presented in [7].

In the motivating example, a set of assets are used among tools. To better support artifact reuse with RAS, asset repositories, also known as reuse repository [28], are required. A repository is recommended to place reusable software assets [16][30]. However, existing asset based repositories are not able to retrieve assets based on a reuse process context, a lack that implies in inconformity between asset dependencies. In Section II A, an example of reuse process pointed that many assets can be composed in alternative ways, dependencies between assets must consider a tailored reuse process to bring assets and respective dependencies as exemplified by those three repositories shown in Figure 2. Thus, the main difference from existing repositories is that the reuser searches for assets and retrieves artifacts plus all RAs related to the selected artifacts considering a tailored process.

The last activity shown in Figure 2 is used to define the context for assets and RAs intending to tailor these elements based on communality and variant designed in a Feature Model [18]. In this sense, all this information must be stored into
specialized RAS based repositories that execute operation to retrieve data based on a selected user context. In this sense, we are extending an asset repository [28] to support contexts related to artifacts.

C. Reuse Assistant Contextualization

This section depicts the proposal for the “Acquisition” step shown in Figure 1. Activities shown in Figure 3 are executed by asset users and also by a model repository in response to user actions. In the first activity, a user searches the repositories for an artifact and receives a context as response in activity 3. As response, the repository must suggest reuse assistants that best fit to support a user’s context (activity 2). After he/she searches for reusable artifacts, the repository would also suggest assets for acquisition/download packed in a RAS Asset File. Thus, to reach this goal, it is required to capture the user’s context as illustrated by activity 3.

A context can be manually provided by (re)users or be automatically captured, because the set of acquired assets can provide information about the user’s intentions. Related works propose solutions to contextualize a user environment, such as [15] and [12]. These solutions can also be used in this research to support a more detailed asset contextualization.

After activity 3 is executed, in activity 4 the repository asserts that the selected context allows to return a set of reuse assistants that match the user context. Finally it tailors assets and packs a RAS file composed by reusable assets and reuse assistants (activity 5). This is a detail that deserves deep discussion because it validates the matching between the selected context and the packaged reusable assets.

Practical Application: Our proposal uses the Feature Model to contextualize reusable assets (see Figure 2 activity 3) as done in other works to solve other problems regarding reuse techniques [4][12]. It is the case for reuse tasks 5 and 7 discussed in Section II.A, where transformation processes/chains has been tailored successfully for a set of MDD tool configurations. Thus, these relationships between context variables can be used to support tasks that only fit to a selected feature model. In other words, associating features with assets allows retrieving assets and all other RAs (e.g., model transformers and wizards) related to those assets in the MDD technique example. This is illustrated by the RAS Asset File elements shown in Figure 3.

D. Reuse Assistant Executions with a Reuse Process

RAS documentation also suggests that reuse activities should be executed and assisted by programs. However, RAS does not support details to specify tasks execution. To overcome this deficiency, RAS suggests extending the metamodel to support details about an asset. Accordingly, our proposed extensions [7] allow linking executable assets with reuse process languages such as RDL (Reuse Description Language) [23] and also a transformation from a RAS model to task execution languages. This is illustrated in Figure 4 as a solution to support the “Transformation” step (see Figure 1).

A reuse assistant does not execute reuse tasks. Differently, it is a specification that is transformed into configuration files that allow execution support through workspaces plugins. Accordingly, the first activity shown in Figure 4 illustrates a transformation that is automatically performed from reusable assets and reuse assistants (retrieved from tasks shown in Figure 3) to configure workspace, guidance and executions.

Figure 3. Retrieving Assets and RA from Repositories

This is illustrated in Figure 4 as a link between RAS Model to Reuse Processes Specified with RDL: With our RAS extension it is possible to specify RAs with a common representation as ensured in the proof of concept [17] by generating RDL scripts. Currently, reuse processes specified with RDL do not allow executing assistants among tools without a reconfiguration/adaptation. In this sense, we will extend RDL to support activities that are linked to RAS elements, allowing the execution of RAs among tools.

Transformation from a RAS Model to Task Execution Languages: The translation from a RAS model to target task execution languages occurs by means of model transformations [22] in activity 1 shown in Figure 4. Such target languages are available in production environments such as Eclipse [10] in support to ANT scripts. Accordingly, proposals to support model-to-model and model-to-text transformations can be used to transform a RAS model into ANT scripts [4][11]. Therefore, our proposal is not to execute RAs, but to generate scripts available in workspaces that support such execution.

E. Ensuring Consistency in Tool Chain

Another need to facilitate the execution of the scenario discussed in Section II.A is related to tool chaining [1] to orchestrate RAs. Accordingly, a tool inclusion in a chain is also dependent on the assistant’s IO parameters that must be interchanged between the corresponding supporting tools. This is subjected to errors, since the absence of a common representation does not allow programs to verify consistencies on bindings between tasks interoperated assets (parameter matching) [4]. Thus, a common representation presented in [7] can be used to fulfill activity 4 shown in Figure 3, by ensuring that tasks interoperated consistent reusable assets (parameters). Moreover, this requires the usage of rules that can be specified.
using a Business Rule Management System (BRMS) [26] to check constraints inside the proposed asset repository.

**Practical Application**: An important work was recently presented that discussed about dynamic reconfiguration of features models using BRMS [26]. Thus, next step will focus in specifying rules to constraint RA compositions regarding a context defined by a feature model.

F. Final Considerations

We developed an Eclipse plug-in to deploy the RAS model into an Eclipse workspace as a solution to activity I shown in Figure 4. It enables the configuration of executable tasks as ANT tasks and Eclipse Mylyn tasks (to support documentation). Besides, by using the deployer plug-in, we successfully executed the first proof of concept [17], leading to conclude that our extension allows representing information to interoperable reuse tasks among tools [7].

IV. Related Work

In this section, we describe existing work on RAs and tool support that promotes execution guidance and asset specification that can help to achieve an automated chain of reuse activities. This work is divided in four main contribution areas: a) asset specification; b) asset repositories; c) asset contextualization; and d) asset execution.

RAS is the OMG proposal to describe reuse assets. Accordingly, some related work focuses on asset documentation and classification to support a better cataloging, search and retrieval from a component repository [15][11] [16][25]. However, we have not found in literature evidence of work that extends RAS to support the specification of reuse task execution. Given that RAs are mostly like executable activities, this work proposed a RAS extension to grant a successful task deployment and consistency for program execution. Thus, our proposal is to formalize assets with deployment and execution support information as a complement to related work.

RAS related work exemplifies scenarios where assets are only documented. Therefore, it requires less formal definition when compared to what is needed to express executable activities. Nevertheless, some RAS proposals are suggestive to describe reuse tasks: 1) Hadji et al. [15] propose a RAS extension to improve user interactions with component repositories, suggesting the acquisition of artifacts according to the user’s context. Our proposal is similar because the developer must acquire a RAS package according to the artifact context defined by a Feature Model [18]. In this sense, despite our work not being focused on contextualized reusable assets as it is required by the exemplified Figure 1, from steps 4 to 6, some PLA related work such as Atkinson et al. [3], Oliveira et al. [13], Gomaa et al. [14], Czarnecki et al. [8] and Antkiewicz et al. [2] and Odyssey [12] are important to contextualize assets using Feature Model; 2) Elgedawy et al. [11] propose a RAS extension to support a better description about SOA reusable components. With more formal specification, SOA components can be acquired and used in the developer’s production environment. However, authors do not present a solution to deploy these assets; 3) HongMin et al. [16] propose a component repository to store, search and retrieve RAS assets with extensions to support better cataloging; 4) Accordingly, Park et al. [25] propose RAS extensions to support cataloging and documentation of reusable assets. All RAS extension proposals were added into the RAS metamodel designed with EMF and complemented our work.

Another class of related work proposes a non RAS compliant approach used to specify and to execute assistants. This means the usage of DSLs to describe reuse processes [23]. Ortigosa and Campo [24] claim that Software Agents are important to describe an execution flow in regards to an assistant for OOFI. This is a reuse technique where reuse assistants are programmed to generate source-code that specializes classes and operations of an O.O. framework. Accordingly, Oliveira et al. [23] propose a language named RDL to orchestrate reuse activities and a tool named ReuseTool to support execution. On the other hand, we are proposing that reuse tasks be specified using a RAS extension in an EMF based solution. Thus, by using model transformations, one can transform a RAS input model, designed with a generated EMF-based Eclipse plug-in, into a RDL program or ANT scripts.

Reuse guidance tools are related to IDEs. In environments such as Eclipse [10] and Netbeans [21], wizards are represented as a set of forms codified in Java that guides a reuser through a predefined sequence of data-collection tasks. Ortigosa and Campo [24] and Oliveira et al. [23] claim that a simpler solution to describe wizards is preferable, suggesting that wizard tasks be specified with reuse process specification languages. These proposals are interesting to describe and to execute a reuse process task. However, as they are focused in task sequences execution, they fail in detailing reuse activities that provide important instruction information along the execution of reuse assistants. On the other hand, approaches based on RAS provide extensive information about guidance instruction, but lack support to execution. Therefore, our proposal is to use the best of both worlds and bring the link between reuse process execution and guidance instruction provided by RAS.

V. Concluding Remarks and Limitations

Reusable assets such as reuse practices documentation, reuse assistants and software assets exist in many reuse techniques but they lack integration capabilities. Our proposal is to aggregate such assets as interoperable reuse assistants. To achieve this goal we have discussed about a set of adaptations required to a tool set. Accordingly, we pointed to practical experiences, some in industry, which provide validity of this proposal. More important, recently, a study [17] ensured that a RAS extension [7] can be used to interoperate some reuse assistants.

Although this study suggests some success towards our proposal; some ongoing work is still being conducted: to support executable tasks as a RAS model it is necessary to check model element constraints. Thus, in activity 4 shown in Figure 3 of our proposed architecture, to ensure asset’s model correctness, we are planning to support constraints in a RAS model by using a BRMS.
ACKNOWLEDGMENT

This work was partially supported by the Brazilian agencies CAPES and CNPq.

REFERENCES


[17] Interchangeable Reuse Assistants - White Paper 01. Available at <prisma.cos.ufrj.br/wct/irawp01.pdf> At 06/02/2013.


