Requirements and Solutions for Tool Integration in Software Test Automation

Bernhard Peischl
Softnet Austria
8010 Graz, Austria
bernhard.peischl@soft-net.at

Rudolf Ramler, Thomas Ziebermayr
Software Competence Center Hagenberg
4232 Hagenberg, Austria
{rudolf.ramler, thomas.ziebermayr}@scch.at

Stefan Mohacsi
Siemens IT Solutions and Services
1100 Wien, Austria
stefan.mohacsi@siemens.com

Christoph Preschern
Ranorex GmbH
8053 Graz, Austria
christoph.preschern@ranorex.com

***Abstract*—In this article, we exemplified today's requirements in integrating test automation tools in terms of three integration scenarios combining industrial strength tools in the area of test management, model-based testing and test executionThe article further sketches solutions for the three scenarios by introducing various integration concepts and by discussing their advantages and drawbacks. Based on successful results we propose a framework for test tool integration.**

Keywords—software test tools; test automation framework; application integration.

I. INTRODUCTION

The current landscape of solutions for test automation is characterized by a large number of heterogeneous commercial and open source tools. Many of these tools are highly specialized solutions for specific aspects of testing, they focus on different technologies, or they have been designed with certain development and test paradigms in mind. Hence, although there is a large variety of specialized test tools for test case generation, test management, test execution, etc., little support for combining the numerous specialized tools to an integrated solution is offered. In practice, thus, engineers bother about interfacing two or more tools at the technical level rather than being able to integrate and enhance these tools to a custom tool chain that meets the needs of a specific project or organization. Furthermore, besides the provision of technical interfaces between single tools, testing activities require automated support for activities that span across several steps in the testing process and link testing with related activities of software development and project management. Especially with model-based testing gaining momentum, integration requirements have notably increased due to the various ways to represent and evolve test cases in combination with artifacts from requirements engineering, design and development.

From the perspective of test tool vendors and solution providers, the situation is characterized by similar challenges. "80% of the effort Automated Software Quality (ASQ) tool vendors spend today duplicates the work of others, recreating an infrastructure to enable testing and debugging activities. Only 20% of their work produces new function that's visible and valuable to testers and developers." [1] Vendors and developers of test tools have recognized the increasing need for integration that allows them to focus on their specific tool competencies, while still being able to offer a comprehensive testing solution to their customers.

The objective of our work, therefore, is the development of integration concepts for test tools that allow connecting tools from different vendors, each specialized on a particular task in test automation, within an extensible test automation framework. In Section 2, we introduce three commercial software test tools from international tool vendors participating in the Softnet Austria Competence Network. Section 3 describes the application scenarios used for exploring the integration requirements. Section 4 summarizes established integration approaches from which we draw in Section 5, where we present and discuss concepts and first solutions. Section 6 summarizes the paper and outlines future work.

II. TEST TOOL LANDSCAPE

To demonstrate and evaluate the proposed integration concepts, we work together with two international companies developing commercial software test tools that, in combination, represent a lateral cut across typical activities in test automation. The following three tools have been involved in the studied scenarios:

- *IDATG* [3] (Integrating Design and Automated Test case Generation) is a tool for generating test data and test cases that has been developed since 1997 by the Siemens Support Center Test in cooperation with universities and the Softnet Austria Competence Network. The IDATG tool supports various approaches for test design and test case generation including equivalence class partitioning, boundary value analysis, cause-effect analysis [2] as well as random and hybrid test case generation [3]. Over the years, the functionality has been continuously expanded and the tool has been successfully applied in numerous commercial and industrial projects within Siemens and by customers such as the European Space Agency ESA. Today, IDATG is a commercial tool offered in combination with the test management solution SiTEMPPO described in the following.
- *SiTEMPPO* [23] is a solution for managing large test case portfolios and related artifacts such as test data, test

 ISBN: 978-1-61208-168-7

results and execution protocols. The tool supports test planning, test case design and specification, the composition of test suites, manual and automated execution of test cases as well as the analysis and reporting of test results [20]. Test management as the coordinating function of software testing interacts with a variety of other development and testing activities such as requirements management, change and defect management and test automation. Hence, the tool has to offer interfaces to a number of related but separate tools for data exchange and synchronization. SiTEMPPO has been developed by an initiative of Siemens Austria. Nowadays, the tool is applied in projects within Siemens all over the world and it is licensed as commercial product for test management on the open market with customers from various industrial domains as well as commercial and public organizations.

- *Ranorex* [24] is a solution for developing and executing automated test cases. The focus of the Ranorex test tool is on the user-friendly capture and replay of robust test scripts building on the accurate recognition and unique identification of user interface elements of applications based on a broad spectrum of different technologies, from C#, VB.NET, WPF, Flex/Flash, to Java and even Qt. The unique strengths of Ranorex's capturing facilities made it a widely recognized test automation tool successfully applied by numerous customers all over the world. The reliable capturing facility allows for an automatic provision of the various elements of the user interface and can thus support the modeling of user interfaces and workflows. Therefore the Ranorex test tool has also been used in a lightweight model-based approach for random test case generation and execution [4]. With the ever increasing variety of user interfaces and the various notification mechanisms in behind, robust replay mechanisms are further an important part in executing and recording the tests being generated.

III. USAGE SCENARIOS AND REQUIREMENTS

In the context of the tools listed above, various usage scenarios have been identified and investigated.

A. Scenario 1: Test Automation and Execution

The integration of executable test cases provided, e.g., as test scripts in a test management environment like SiTEMPPO is a vital part of automating the test process. In this scenario, we do not address the generation of test cases but take care of the task of executing the test cases (no matter where the test cases stem from) and recording the results in a test case management tool. This scenario involves several tasks. Typically, for every test case we have to provide test data and the path to the test script for executing the test case. The result of the execution is typically persisted in form of a log file. The test management tool has to access and interpret the log file in order to derive the results of the test case execution.

Although a technical solution for interfacing the tools in this basic scenario can easily be envisioned, when coupling tools of two different vendors, a couple of challenges are involved. For example, how can the message *"testscript foo.bar failed in line 42"* be mapped to a step in the test case specification? What is reported if the test case execution is not terminating or terminates with a timeout? Who should be notified when the test execution failed due to a problem in the setup of the execution environment? Such questions are typical for any integration scenario and illustrate that the various aspects involved have to be addressed at different levels of integration by different integration concepts.

B. Scenario 2: Model Evolution in Model-based testing

Model-based testing promises to offer solutions to many of the problems that make software testing a complex task. In theory, given a suitable behavioral model of the SUT, any number of test cases can automatically be generated with respect to planned adequacy criteria and the model serving as a test oracle [7]. To leverage the full potential of test case generation, a complete, detailed and correct model of the SUT – a golden model – has to be provided. Ideally, such a model of the SUT is built on the grounds of requirements or existing specification documents. So the model encodes the intended behavior and can reside at various levels of abstraction [7]. Further models that focus on the workflow and the possible user interactions (e.g., via the GUI elements) may assist in the systematic design of test cases respectively in their automated generation.

Even with considerable upfront investments in terms of resources, time and money, such a golden model can hardly ever be achieved in practice due to several reasons. First, the model needs to capture specific aspects of the SUT at a very detailed level, e.g., GUI elements and workflows. However, in many cases the requirements do not contain the necessary details and, thus, the only options are making adequate assumptions or reverse engineering these missing details by exploring the actual implementation. Second, like programming, modeling is an error-prone task and without frequently executing the model throughout model development, faults in modeling are rather the rule than the exception.

Executable models are known to improve the situation, but are not able to overcome this problem fully. Therefore, tools such as IDATG propose the combination of model-based testing with GUI exploration and capturing techniques employed within capture and replay tools like the Ranorex Studio. This allows for an early detection of faults in the models being developed as test cases, as they can be executed on the GUIs and workflows even in early stages of development when almost no business logic is implemented behind the GUIs. An agile development process, where GUIs - from the very beginning - are crucial elements and are thus directly influencing the modeling process, increase the chance that the software finally will solve the problem of the customer rather than conform to a specification that does not capture the problem in its full shape.

Figure 1 illustrates a scenario for an integrated tool chain. The scenario involves several tools: the Siemens IDATG test case generator, a model editor (e.g., a workflow editor or a UML modeling tool, the IDATG tool comes with its own model editor), the Ranorex GUI spy and the Ranorex replay component. The scenario starts with capturing a specific

 ISBN: 978-1-61208-168-7

view of an application (view 1) and continues with recording of a second view of the SUT (view 2). Capturing of the GUIs establishes a rather detailed level of modeling from the very beginning when compared to a purely manual modeling process. The process of capturing introduces conceptual units and recurring building blocks, which support reuse between test cases and even between test cases across different projects.

Afterwards, the result is handed over to a modeling tool where the result of the recording process (view 1 and view 2) is combined and enriched with further details from the requirements document or the knowledge of the SUT. The automated extraction of model components alongside with the composition of components reduces the upfront investments and thus removes a substantial entry barrier into model-based testing also from an economical perspective.

Thereafter the criteria for the test case generation are specified and the model is handed over to the IDATG test case generator for generating the test sequences (which correspond to paths in the model) and corresponding test data. Finally, the Ranorex replay component is employed to execute the generated test cases on the GUI of the SUT.

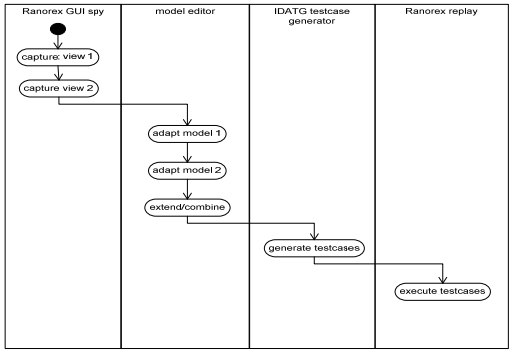

Figure 1: Example workflow with an integrated tool chain.

C. *Scenario 3: Managing Requirements-based Testing*

Testing that the specified requirements have been correctly and completely transferred into executable software is an essential part in the software development lifecycle. In this scenario testing embraces a range of verification and validation activities as well as interfaces linking the results to development and management. In particular, this scenario demonstrates the need to integrate tools across the test and development process to establish a tool chain where the results of one phase build the basis for the next phase. However, the integration is not only characterized by passing on results but includes several update and feedback cycles.

SiTEMPPO supports a requirements-based approach for testing by organizing the test case portfolio according to the structure of the requirements, by tracing test cases to requirements and by reporting test results from the perspective of covered requirements. Figure 2 gives an overview of the involved activities and interactions.

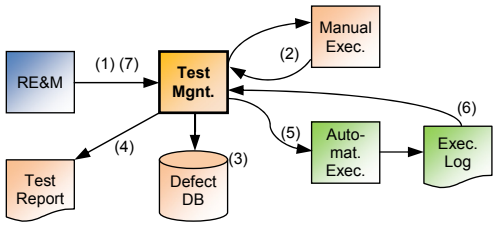

Figure 2: Activities and interactions in requirements-based testing.

(1) Requirement trees are imported into the test management tool as read-only structure. For every imported requirement one or more test cases are derived. The tree structure is used to organize the set of new test cases. Coverage reports show which test cases are linked to requirements and, vice versa, which requirements are covered by test cases.

(2) In a first run, the test cases are executed manually. The test execution results are evaluated and (3) defect reports are issued to a separate defect database when bugs are encountered. (4) Furthermore, the evaluated test execution results are mapped to requirements, indicating that the implementation of a requirement has either been successfully verified or still contains bugs. This first manual run has a strong explorative character and not only focuses on testing the software system but also serves as check whether the requirements have been correctly translated into test cases.

(5) For stable requirements that are subject to ongoing regression testing, test engineers – often located at distributed development sites – automate the manual test cases with tools such as Ranorex Studio or IDATG. The resulting test scripts are linked to the test cases in the test management tool. As described in Scenario 1, SiTEMPPO provides mechanisms for running the test scripts from within the test management environment and (6) for collecting the execution results to evaluate which test cases passed or failed. The results are again mapped to requirements for reporting.

In many projects changing requirements are a constant factor that adds further complexity and dynamics to requirements-based testing. (7) Changes in the requirements have to be propagated to the derived test cases and, furthermore, to the associated test scripts. Keeping requirements, test cases and test scripts synchronized requires coordination and collaboration between the different roles such as requirements analyst, test manager and test engineer. However, without appropriate mechanisms incorporated in test and development tools, coordination and collaboration becomes an ever increasing challenge for distributed teams.

While most of today's tools lack support for coordination and collaboration, SiTEMPPO already includes basic mechanisms like versioning of test cases and linking execution results to the corresponding version of a test set. Nevertheless, as users demand short feedback cycles and constantly up-to-date information on the status and progress of testing across all involved roles and activities, future solutions need to close the currently existing gap between the different tools at the process level.

 ISBN: 978-1-61208-168-7

Layer	Technology	Communication	Coupling	Interaction dynamics	Transformation	Usage context
Business process	**Workflow Engine**	both	transparent	business rules	low	multi role
Application	**Service Bus**	both	transparent	business rules	yes	multi application
	Messaging	asynchron	transparent	registration	low	multi application
	Service	synchron	loose	registration/broker	low	single application
	Business components/RPC	both	strong	static	low	single application
	Shared library/API	synchron	strong	static/plug-in	low	single application
Data	**Database**	both	strong	static	low	single/multi app.
	File	both	strong	static	possible (XSLT)	single application

TABLE I. OVERVIEW OF INTEGRATION CONCPETS

IV. INTEGRATION CONCEPTS

The area of Enterprise Application Integration (EAI) has a long history in developing integration concepts for interaction between existing functionality. Approaches for integration can be categorized by the architectural level where the integration is established [11] or by the communication paradigm underlying the integration [12]. We adopted the categories proposed in literature and summarized the existing integration concepts in Table 1.

In Table 1, the column *Layer* indicates the architectural layer at which the integration is taking place. *Technology* names the commonly applied technologies used for integration. *Communication* shows whether the possible communication options are synchronous, asynchronous or both. The column *Coupling* indicates the strength of the connection between the integrated applications. *Interaction dynamics* states whether the integration is static or dynamic, i.e., has to be set up before the start of the application or can be established and changed at runtime. Data *Transformation* is an important aspect for data exchange between applications and is therefore supported by some of the listed integration concepts. *Usage context* indicates from the user perspective whether integration is possible with one or more other applications.

In the following, the integration concepts as presented in Table 1 are briefly described.

- **File** and **Database:** Integration at the lowest architectural level, the data level, allows the exchange of data between otherwise heterogeneous applications. Data level integration can be implemented in various ways, e.g., by file exchange, by sharing a database, or by copying data from one database to another [15]. This approach may include data transformation if data structures are not compatible. File data might be structured as XML data which provide a stable basis for data exchange and transformation, e.g., using XSLT. While the communication at data level is often easy to implement and has minimal impact on the existing applications, the main drawback of this level is that the applications' existing functionality is not integrated and therefore not reused. Redundant implementations of the same functionality may lead to an increased development and maintenance effort and, furthermore, increases the risk of incompatibility between applications.
- **Shared Library** and **Application Programming Interface (API):** Good software design encourages the reuse of existing implementations, e.g., provided as components in a shared library or in form of plugins. Interfaces encapsulate the functionality and implementation. Via interfaces the functionality of other applications can be accessed. Integration at application interface level (see [11]) can be implemented at different abstraction levels like integration of data access functionality or integration of functionality that contains business logic. Integration at this level leads to strong coupling between applications. Transformation is not supported by default and it supports integration with a single other application. However, if an application already provides an API, implementing integration at this level is easily achievable even without additional infrastructure.
- **Business components, Remote Procedure Calls (RPC):** At higher abstraction levels an application may consist of business components that provide rather coarse-grained business functionality [13]. This functionality can be integrated in other applications in various ways, either by packing them to the application where they should be integrated or by remote procedure calls. Using business components remotely requires that the remote application is running. Business components provide the highest functional abstraction level of an application and, therefore, reuse at the highest functional level. Coupling at this level is strong and transformation support not natively built in. Yet the functional reuse level is high.
- **Service:** Software services provide means for loose coupling of applications as they encapsulate functionality and the site where this functionality is running [14]. The concept of Web services provides standardized protocols for communication to integrate applications across platform borders. Overall, integration at service level means integration at a coarse-grained business function level for reusing application functionality at business level. The advantage of this level is the loose coupling and mostly standardized communication protocols, but without additional infrastructure, communication is still synchronous without transformation support and it is used for integration with a single application.
- **Messaging:** In some cases asynchronous communication is required due to performance reasons or the need

 ISBN: 978-1-61208-168-7

for weak coupling. These requirements can be addressed by a message queue which decouples communication partners in a timely manner and provides guaranteed message delivery. Message queues are also used for sending messages to multiple applications (broadcast), with or without feedback about delivery; see integration styles in [6]. Advantages of this integration level are asynchronous communication, low coupling and integration possible with multiple applications.

- **Service Bus:** A common integration concept is the service bus which provides functional support for the integration and communication between applications. The idea is to connect all applications with a bus where applications put messages on the bus and others listen and take the messages relevant to them. A service bus also supports plugging in additional components like transformation or filter components that allow modifying or removing messages. Furthermore, some service bus implementations support defining message flows between applications and components including splits and joins [6]. This integration level supports all features presented in Table 1 except the possibility of integration along a workflow involving responsibilities and roles.
- **Workflow Engine:** From a user perspective, the usage of applications follows organizational workflows which define task order and responsibilities. In order to accomplish the work, a workflow might contain multiple tasks that utilize different applications. From a technical perspective, a sequence flow between tasks utilizing different applications indicates integration of those applications (see also [16]). Workflow engines are able to implement communication at workflow level and coordinating the use of applications integrated at a technical level. This is the highest and most abstract integration level with support for transparent coupling, dynamic interaction based on business rules and integration of the work processed by multiple roles.

V. SOLUTIONS AND DISCUSSION

This section describes and discusses how the integration requirements elaborated from the usage scenarios in Section III can be supported by the technologies presented in Section IV. The integration concepts have been explored either via a (prototypical) implementation or a design study elaborated together with developers and architects of the test tools.

A. Scenario 1: Test Automation and Execution

In coupling the SiTEMPPO test management solution with the Ranorex test automation tool, we follow the paradigm of a strong coupling with the need for both, asynchronous and synchronous communication. Due to the specialized interface, the interaction dynamics remains static without the need for transformations. Thus, the integration is established via files, i.e., at the level of the data layer (Table I). In detail, the prototypical integration of the SiTEMPPO and Ranorex tools has been implemented as follows:

Ranorex Studio allows creating executable test suites. When the execution of a set of automated tests is triggered in SiTEMPPO, Ranorex Test Runner is called for each test case, passing the name of the corresponding test script as command line parameter. The execution generates a log file in a predefined directory, which is processed by an import adapter implemented as part of SiTEMPPO. The adapter extracts the information relevant for deciding on the test result (passed, failed or blocked).

In order to access the Ranorex tool from within SiTEMPPO, several global settings have to be made in the configuration of the test management environment, e.g., the path to the executable test scripts, the execution log, and the necessary runtime libraries. Hence, the interface implementation part of SiTEMPPO requires exception handling strategies to deal with erroneous configurations and timeouts. Additional setup, rollback and restart mechanisms need to be included in the automated test scripts. Furthermore, predefined execution orders due to implicit dependencies between test scripts cannot be handled by SiTEMPPO.

The benefit of the low-level, static coupling between the two tools is the straightforward implementation of the interface and the ability to consider tool-specific extensions. This benefit turns into a drawback as soon as interfaces for several different test automation tools should be provided. Developing and maintaining a large set of interfaces is cumbersome as the external interfaces may change without notice whenever a new version of an integrated tool is released.

Our experience with implementations for this scenario showed that the initial use case also stretches into the organization dimension. While in an ideal setting the test management supervises the top-down development of automated test scripts from previously defined and specified test cases, in practice, many valuable test scripts also emerge bottom-up and need to be incorporated into the managed test structure. Gathering existing test cases and keeping them synchronized results in a considerable effort for test managers, especially in a distributed project setting. Hence, the need for tool support for discovering and "importing" existing test cases soon appeared as additional requirement. As a consequence we propose an approach emphasizing the inversion of control – developers of automated test scripts should register the new or changed test cases with test management. The responsibility to maintain and update the test cases remains with the test script developers. Integration concepts that support this approach are presented and discussed as part of Scenario 3, Section C.

B. Scenario 2: Model Evolution in Model-based Testing

A key requirement for our Scenario 2 is the interaction dynamics. Any solution has to guarantee acceptable response times and ease of use in switching from one tool to the other. Thus, we favor synchronous communication mechanisms and no or rather low need for transformations. There are no multiple roles involved and the interaction happens always between two tools. Thus we propose shared libraries, business components, and plug-ins to implement Scenario 2.

Plug-ins are a common mechanism for adding third-party tools to a tool suite. A plug-in explicitly provides information about its dependencies on other plug-ins. Furthermore, a plug-in can change menus and menu entries as well as popup menus and toolbars. Additionally, it is possible for plug-ins

 ISBN: 978-1-61208-168-7

to notice the execution of menu actions of other plug-ins [8]. For these reasons, a plug-in mechanism is very well suited to implement the desired coupling on the application level.

According to [9], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. Besides the specification of provided interfaces, the definition of a component also requires components to specify their needs. In other words, a component requires a specification of what the deployment environment will need to provide such that the component will operate. In principle, this is a generalization of plug-ins and might thus be appropriate for implementing the coupling as well.

Software engineers in general create many scenarios (and corresponding model artifacts) and often recall prior work as they develop models for novel use cases. The process of refinding patterns is a popular approach in this respect and will be supported by the concept of a shared library. However, the adequate abstraction level of the models (or building blocks) stored in the library is a challenging research issue. Basically, we pursue two main directions in supporting reusability: tagging and structural similarity [5].

C. Scenario 3: Managing Requirements-based Testing

The third scenario is characterized by the need for integration at the process level to support coordination and collaboration across different roles, phases and distributed development sites. Conventional approaches rely on a central coordination instance, usually represented by test management. In that constellation the test management tool is used as central hub, gathering and consolidating information from the various other test tools. Technically, the interfaces between the involved tools remain on the lowest level; mainly data exchange via import/export facilities is supported.

The specialization of the different tools is generally quoted as reason why sharing functionality between tools is insufficiently attractive. However, the numerous redundant features provided by the different tools reveal that the opposite is true. For example, almost all tools implement their own reporting. The slight but obvious variances in the reporting of the different tools are a common nuisance for users, especially when they try to analyze the status of testing over all activities from data spread across different tools. As a result, existing reporting facilities are once more implemented as part of test management tools in an attempt to create a homogeneous, aggregated view on the test process.

With the test management tool as central hub and all other tools arranged as satellites, the management tool becomes the bottleneck in the test tool infrastructure. It has to provide interfaces to all tools included in testing and, thus, the provided interfaces are the main limitation in the choice of applicable tools. Projects suffer from this inflexibility when the optimal test tool cannot be applied due to test management not offering the corresponding interface or – in case generic adapters exist – when test management lacks the resources to setup and maintain the necessary interface configurations. Moreover, the strong coupling of the data level integration turns intro rigid dependencies. Even minor changes in the data format may render the interface incompatible. Hence, in practice, many projects are tied to outdated versions of tools because of update incompatibilities. Tool providers, however, often do not even know about the potential conflicts since they are not aware of the dependencies to the interface implemented as part of the test management tool.

As indicated in Scenario 1, Section A, we propose to emphasize the Inversion of Control principle for tool integration at the process level. Test management has to be released from the burden of gathering and extracting data from the various other test tools. In contrast, the satellite tools have to take over the responsibility of providing the necessary data and maintaining compatibility. Now, however, instead of test tools interfacing directly with various different test management tools resulting in a complex point to point integration, the tool communication should be extracted into a separate integration facility serving as backbone of the tool infrastructure. Service-oriented concepts have been proposed and were successfully evaluated for software engineering environments [17]. Drawing from positive experience with integrating software engineering tools, we adopted the service bus approach (Figure 2) specifically for test tools.

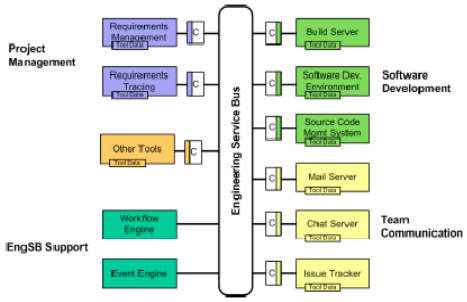

Figure 3: Engineering Service Bus (EngSB) for integrating software engineering tools [17].

The illustrated approach enables communication between the different tools beyond the level of data exchange. Status messages can be exchanged to notify other tools about completed activities and pending updates. For example, test execution tools can send a message indicating the successful completion of a test run. The message can include relevant result information and a link to the execution log. Thus, instead of storing static configuration details such as the location of execution logs in the test management tool's settings, all concerned tools register for the corresponding message and receive the information at runtime. Furthermore, the link may not point to a static location from where the log is retrieved as file, but to a service interface that allows querying and analyzing relevant aspects of the execution. Providing the query logic as a service of the execution tool avoids redundant implementation of analysis functions.

A prerequisite for the service-based integration of tools is the agreement about offered services, data structures and exchange formats. In software and systems engineering and in particular in testing, several relevant standards are in place, for example the UML Testing Profile [21, 22], the IEEE Std. 829-2008 for Software and System Test Documentation, or the Requirements Interchange Format [18].

 ISBN: 978-1-61208-168-7

Furthermore, automated transformation of messages, models and data formats implemented in form of services are also connected to the service bus.

Communication and teamwork requirements can be addressed by adding shared services for reporting, monitoring, status notification and even workflow-based collaboration. An example for a tool providing shared services is a test cockpit [19] providing insight on the status and progress of testing across all involved roles and activities.

VI. CONCLUSION AND FUTURE WORK

In this article, we exemplified today's requirements in integrating test automation tools in terms of three integration scenarios combining industrial strength tools in the area of test management, model-based testing and test execution: The test and requirements management tool SiTEMPPO, the Siemens IDATG tool for model-based testing, and the Ranorex automation tool suite. The integration scenarios represent typical situations frequently encountered in real-world projects by the authors: (1) Combining test automation and test execution, (2) model development and evolution in model-based testing, and (3) the management of requirements-based testing and regression testing. For each of these scenarios, solution concepts have been developed and explored together with developers and architects of the presented tools, based on existing integration technologies (file-level data exchange, plug-in concept, messaging and service bus). It could be shown that the elicited integration requirements of each scenario can be addressed by applying existing concepts, which are attributed the potential for building a framework able to combine a set of heterogeneous tools by different vendors. Although the higher-level integration concepts show a larger potential w.r.t. integrating heterogeneous tools, we also found that no single integration concept is able to cover all requirements from the explored scenarios.

Our next step will be to consolidate the existing implementations and concepts towards a service-oriented integration platform easily extendable by future test and development tools.

ACKNOWLEDGMENT

The research herein is partially conducted within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

REFERENCES

[1] The Hyades Project Automated Software Quality for Eclipse: http://www.eclipse.org/tptp/home/archives/hyades/project_info/HyadesFormation.12.pdf, last visited on 27th June 2011.

[2] A. Beer and S. Mohacsi, "Efficient Test Data Generation for Variables with Complex Dependencies", Int. Conf. on Software Testing, Verification, and Validation, 2008, pp. 3-11.

[3] S. Mohacsi and J. Wallner, "A Hybrid Approach for Model-Based Random Testing", in Advances in System Testing and Validation Lifecycle (VALID), 2010, pp.10-15, 22-27 Aug. 2010

[4] B. Hofer, B. Peischl, and F. Wotawa, "GUI Savvy End-to-End Testing with Smart Monkeys", Fourth International Workshop on the Automation of Software Test, Vancouver, Canada, May 16-24, 2009.

[5] W.N. Robinson and H.G Woo,"Finding reusable UML sequence diagrams automatically", IEEE Software, vol. 21, no. 5, pp. 60- 67, Sept.-Oct. 2004.

[6] G. Hohpe and B. Woolf, "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions", Addison-Wesley Professional, 2003.

[7] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches", Software Testing, Verification and Reliability, 2011. Published online, paper version in press.

[8] S. Burmester et al., "Tool integration at the meta-model level: the Fujaba approach", Int. J. Softw. Tools Technol. Transf. 6, 3 (August 2004), pp. 203-218.

[9] C. Szyperski, "Component Software: Beyond Object-Oriented Programming", 2nd ed. Addison-Wesley Professional, Boston ISBN 0-201-74572-0.

[10] Eclipse TPTP, Eclipse Test & Performance Tools Platform Project,http://www.eclipse.org/tptp/, last visited 27th July 2011.

[11] D.S. Linthicum, "Enterprise Application Integration", Addison-Wesley Professional, 1999, ISBN.: 978-0-201-61583-8.

[12] Gregor Hohpe, Bobby Woolf: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, 2003.

[13] P. Herzum and O. Sims, "Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise", John Wiley & Sons, New York, NY, USA 2000, ISBN:0471327603.

[14] T. Erl, "Service-Oriented Architecture: Concepts, Technology, and Design", Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[15] M. Vujasinovic and Z. Marjanovic, "Data Level Enterprise Applications Integration", in Business Process Management Workshops, pp. 390-395, Volume 3812, Lecture Notes in Computer Science 2006, Springer.

[16] J.A. Espinosa and A. Sanz Pulido, "IB (Integrated Business): A Workflow-Based Integration Approach", Hawaii International Conference on System Sciences (HICCS), 2002.

[17] S. Biffl and A. Schatten, "A Platform for Service-Oriented Integration of Software Engineering Environments", 8th International Conference on Software Methodologies, Tools and Techniques (SOMET 09), 2009.

[18] M. Jastram and A. Graf, "Requirements, Traceability and DSLs in Eclipse with the Requirements Interchange Format (RIF/ReqIF)", Dagstuhl-Workshop MBEES 2011: Modellbasierte Entwicklung eingebetteter Systeme, 2011.

[19] S. Larndorfer, R. Ramler, and C. Buchwiser, "Experiences and results from establishing a software cockpit at BMD Systemhaus", 35th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA 2009), pp. 188-194, IEEE, 2009.

[20] R. Ramler, G. Czech, and D. Schlosser, "Unit Testing beyond a Bar in Green and Red". 4th int. Conf. on Extreme Programming and Agile Processes in Software Engineering, XP 2003.

[21] OMG, "UML testing profile Version 1.0", OMG, 2005. formal/05-07-07; http://utp.omg.org/.

[22] P. Baker, Z. Ru Dai, J. Grabowski, O. Haugen, I. Schieferdecker, and C. Williams, "Model-Driven Testing: Using the UML Testing Profile", Springer, 2007.

[23] SiTEMPPO: www.siemens.at/sitemppo, visited 27th July 2011.

[24] Ranorex Automation Studio: www.ranorex.at, visited 27th July 2011.

.