Towards a Maintainable Federalist Enterprise Measurement Infrastructure

Matthias Vianden, Horst Lichter, Andreas Steffens
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
{vianden, lichter, steffens}@swc.rwth-aachen.de

Abstract — **Large scale measurement systems are hard to build and to maintain. In this paper we propose an architecture blueprint for a federalist Enterprise Measurement Infrastructure (EMI) which helps to address these typical weaknesses of centralistic measurement systems. The EMI is based on the ideas of Service Oriented Measurements. We combined these with modern ideas from the area of Enterprise Application Integration and extended the ISO 15939 data flow to allow a more flexible and elegant solution. The current prototypes of EMI implementations and field studies prove the benefits of the architecture blueprint over existing solutions. We strongly belief that the EMI can help to build better, extendible, and maintainable measurement systems which are integrated and aligned with modern business needs.**

Keywords — Measurement Infrastructure; Enterprise Application Integration; Metric; Dashboard; Service Oriented Architecture

I. INTRODUCTION

Software metrics are an important means to measure the quality of both the development processes and software systems. Improvement reference models such as CMMI require that software development organizations build up abilities to systematically apply metrics to support project management [1]. Based on quantifiable metrics process managers are able to identify processes that contribute to project success or failure. Hence, metrics are a necessity for objective process optimization. However, it is often difficult to integrate measurement values from a large variety of different software systems used in software development projects.

Resulting in the different application scenarios for dashboards and measurement systems (strategic, analytical, or operational [2]) modern measurement systems use new integration approaches. Most recently considerable research was devoted to using service oriented (SOA) and agent based architectures for measurement systems [3]. New loosely coupled integration architectures are researched in the area of enterprise architecture integration (EAI) [4], [5], [6], [7], [8], [9]. Unfortunately, these ideas are not systematically used in measurement infrastructures. Most of the solutions found in the industry right now are based on BI (Business Information) systems. However, all of the proposed solutions (even the new SOA and agent based approaches) use a central database or system to store and integrate the measurement data. Hence, they suffer from well known centralized integration problems; like the need for a common data schema to integrate different applications. Additionally, not every data should or can be measured (and stored) by means of relational data schemata [10].

In this paper we propose the central requirements for a loosely coupled architecture blueprint of an Enterprise Measurement Infrastructure (EMI) that is aligned with the needs of the different measurement stakeholders as well as the ISO 15939 measurement model. Additionally, the proposed infrastructure uses federalist data storage and measurement systems to overcome the weaknesses of a central measurement repository.

The following chapter II introduces a typical application scenario for an enterprise measurement infrastructure. This is based on an example taken from our industrial experience. Based on this example we derive three distinct stakeholders with different requirements to the infrastructure in chapter III. Based on this we investigate existing integration and measurement infrastructures in part IV. Chapter V explains in detail the architecture and components of the proposed federalist enterprise measurement infrastructure. Chapter VI provides first experiences with the infrastructure. The paper is concluded in chapter VII.

II. APPLICATION SCENARIO

Project managers and other cross sectional roles in large organizations typically require a lot of information from different systems. For example they are required to control the budget, schedule, costs, quality, requirements, tests, risks and so on. All these control tasks lead a large variety of information needs for the project manager. The answer to a single information need can often be derived from a set of information stored in a certain repository. Different types of information are typically stored in different repositories and often even the same type of information is stored in various repositories. From our experience this problem is not only limited to large organizations. Even project managers in medium sized organizations may need to look into five different Change Request Management (CRM) systems (product development, service, custom solutions, and customer issue tracking (products and custom solutions)) to gather all relevant information.

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

The following example is used to understand the requirements from a business point of view and aligns the different parts of the enterprise measurement infrastructure. We assume a company that established an ISO 15939 compatible measurement process. In this company projects typically last for a year. A project manager needs to keep a close eye on the budget of the project, the schedule, the overall quality and the risks of the project. She likes to utilize a measurement dashboard that provides charts for the development of the measurers per month.

The base data for the dashboard is stored in different systems and the data derived from the systems is not uniform. The initial budget (measured in person days) and schedule (measured in calendar days) is stored in a procurement management system per work package. Hence, the budget for the complete project is a sum of all the budget of all work packages assigned to the project. The overall schedule can be derived from the work packages as well. The project utilizes a Change Request Management (CRM) system to store the requirements which are associated to work packages. This system also keeps track of the working hours per requirement. The sum of all the time spend on the project can therefore be derived from the work packages in the CRM system. The current schedule can be derived from these work packages as well using a project management tool. The overall quality of the project is monitored using a tool for static source code analysis that measures metric and rule violations. Additionally, the amounts of errors in the system (which are stored in a different CRM system) are used to monitor the overall quality as well. The project manager might want to filter the errors according to their priority (only high priority errors or all errors). The risks are stored in a spreadsheet which is updated regularly and when addressing certain risks.

This very simple scenario already shows five different systems (procurement management system, project management tool, task CRM system, error CRM system, risk spreadsheet) that need to be integrated to provide measures for the dashboard.

III. Requirements for an Enterprise Measurement Infrastructure

The measurement infrastructure needs to address the needs from different stakeholders. After carefully examining the literature and based on our experience we identified the following three main stakeholders:

- Measurement Customer
- Developer
- Operator

Each of these stakeholders provides a unique and specific set of requirements regarding the architecture and the provided functionality of the underlying measurement infrastructure. First we investigate the requirements of the measurement customer which holds the main set of functional requirements. Later the developer and operator roles add mainly non-functional requirements for the measurement infrastructure.

A. Measurement Customer

A project manager is a typical example of a measurement customer. She is interested in the actual status of her project and does not care (and should not!) about the way the data is collected or metrics are calculated. Like in the example above, measurement customers have a brought variety of information needs. Unfortunately, the answers to the different information needs are stored in various repositories. The scenario introduced above includes different systems for budget, scheduling, tasks and risk information. The resulting central requirement of the measurement customer for an EMI is the integration of these systems in a way that a comprehensive calculation of metrics is possible.

To achieve this goal, the infrastructure has to cope with the heterogeneity of those systems. Heterogeneity appears on various levels of a software information system. Wache et al. [11] define structural and semantic heterogeneity of data. Structural differences lead to the problem of schema-mapping, a quite well known field of research in the database community [12]. Hence, data heterogeneity is challenging for a successful integration of those systems as well.

Additionally, measurement customers demand correct and up-to-date data because old or incorrect data lead to wrong conclusions and wrong decisions. Hence, an EMI should provide mechanisms that guarantee a fast recognition and processing of relevant events inside the system landscape. Additionally, this requires a robust and highly availability infrastructure.

Our experience with many industry partners shows that the information needs of measurement customers often change over time. For example development tools and systems are replaced by other tools or systems (tool evolution). Of course, the new tools and systems need to be integrated in the infrastructure. Additionally, processes and organization schemas of enterprises often evolve as well. Especially reorganizations lead to new and changed responsibilities of individual measurement customers and roles which inevitably lead to changes in information needs. Concluding from this, an important requirement for an enterprise measurement infrastructure is to support the evolution of metrics, integrated systems, and visualizations.

B. Developer

The developer needs to implement metrics, visualizations and tools to gather data. The infrastructure needs to support the developer with a clear structure and concepts for all specific tasks. The task to integrate a new system into the infrastructure to gather its data is completely different from the implementation of a new metric calculation algorithm or the implementation of a new visualization. Hence, a requirement for the infrastructure is the clear separation of system integration, calculation, and visualization.

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

C. Operator

This role is often ignored while building and conceptualizing a measurement system or measurement infrastructures. The operations department has two main responsibilities. First, it has to guarantee that all systems are working inside their operational parameters. This requires a dedicated set of operation tools as part of the infrastructure. The infrastructure should at least provide or support a monitoring tool which allows analyzing the amount of data that is transported and stored in the infrastructures components. Second, the operations department has to solve upcoming problems in the infrastructure without disturbing the integrated systems as these systems are often of crucial importance for the company. The operation department is also responsible for the alignment of the system landscape of the organization. Hence, the infrastructure should be compatible with service oriented architectures found in modern organizations.

D. Requirements

The sections above motivate the following main functional and non-functional requirements for an enterprise measurement infrastructure:

R1. Integration of heterogeneous systems to provide the basis for different metrics and visualizations.

R2. Fast and up-to-date recognition and update of the metrics on a change in an integrated system.

R3. Clear separation of system integration, calculation and visualization.

R4. Be robust to avoid a complete system failure if a small part of the system fails. Additionally, the failure of the infrastructure should not result in a failure of the integrated systems.

R5. No central database to store the measurement values.

R6. No central data schema to avoid schema-mapping problems.

R7. Support evolution of metrics, integrated systems, and visualizations.

R8. Offer dedicated operation tools.

R9. Be compatible to Service-Oriented-Architectures.

This set of requirements will inevitably lead to a loosely coupled federalist infrastructure.

IV. Existing Integration Architectures and Measurement Infrastructures

The integration of heterogonous data sources is the main requirement of an enterprise measurement infrastructure. Throughout the last decade several approaches have been proposed to deal with the emerging problems faced by developers and architects of dealing with heterogeneous systems and software landscapes. In the following we present and analyze four existing types of enterprise application integration approaches and distributed measurement infrastructures based on the proposed set of requirements.

A. File based integration

The simplest integration approach uses files to exchange data between applications. One application exports the data needed by another as a dedicated file. This file is imported by the other application and processed. This form of integration has certain disadvantages. Most importantly there is no real communications between the applications. Additionally, the export and import of data has to be synchronized. This directly violated the requirements R1, R7, R8, R9, and most importantly R2.

B. Common Database

This integration approach uses a common database for all integrated systems. This provides a low latency to recognize and process relevant events. The main drawback is the necessity of an additional database management system. Successful implementations of this integration type are data warehouse systems. They provide an integrated database organized in a star schema [13], which includes multi-dimensional aggregated data cubes.

Integration via a common database or a data warehouse is the most common used integration approach chosen by recent measurement systems like Rational Insight. Hackystat[1] and **sonar**qube[2] are other examples for measurement tools that use a common database to integrate measurement data. However, as these systems are based on centralized data bases they directly violate requirement R3. Additionally, they are not able to guarantee requirement R2 and R7.

C. Service Oriented Architecture

Service-Oriented-Architecture (SOA) also offer a common used infrastructure pattern for integration solutions. Systems following the service-paradigm [14], [15] provide a stable self-describing interface for accessing internal data and functionalities. Web services are a prominent example for service-based software systems, which uses XML or JSON over HTTP for their communications. Kunz et al. [16] propose a measurement infrastructure based on SOA-inspired service center. Even thought SOA based integration is quiet common a pure SOA solution typically does not provide the required set of operation tools. Hence, this violates requirement R8. Additionally most of the integration solutions use SOA only for the communication between the visualization clients and a central data base server which violates requirement R5 and R6.

Enterprise Information Integration (EII) is a special case of SOA based enterprise application integration. The main goal of EII is to avoid a central database [17]. EII adds a central query processor to an infrastructure of loosely coupled services. This central processor divides a query into sub queries to the services and aggregates their results. Even though this is an elegant solution to avoid a central database it violates requirement R4 and since the central processor needs to wait for all the sub queries to finish before returning an aggregated result it can take a while before the system

[1] https://code.google.com/p/hackystat/
2 http://www.sonarqube.org/

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

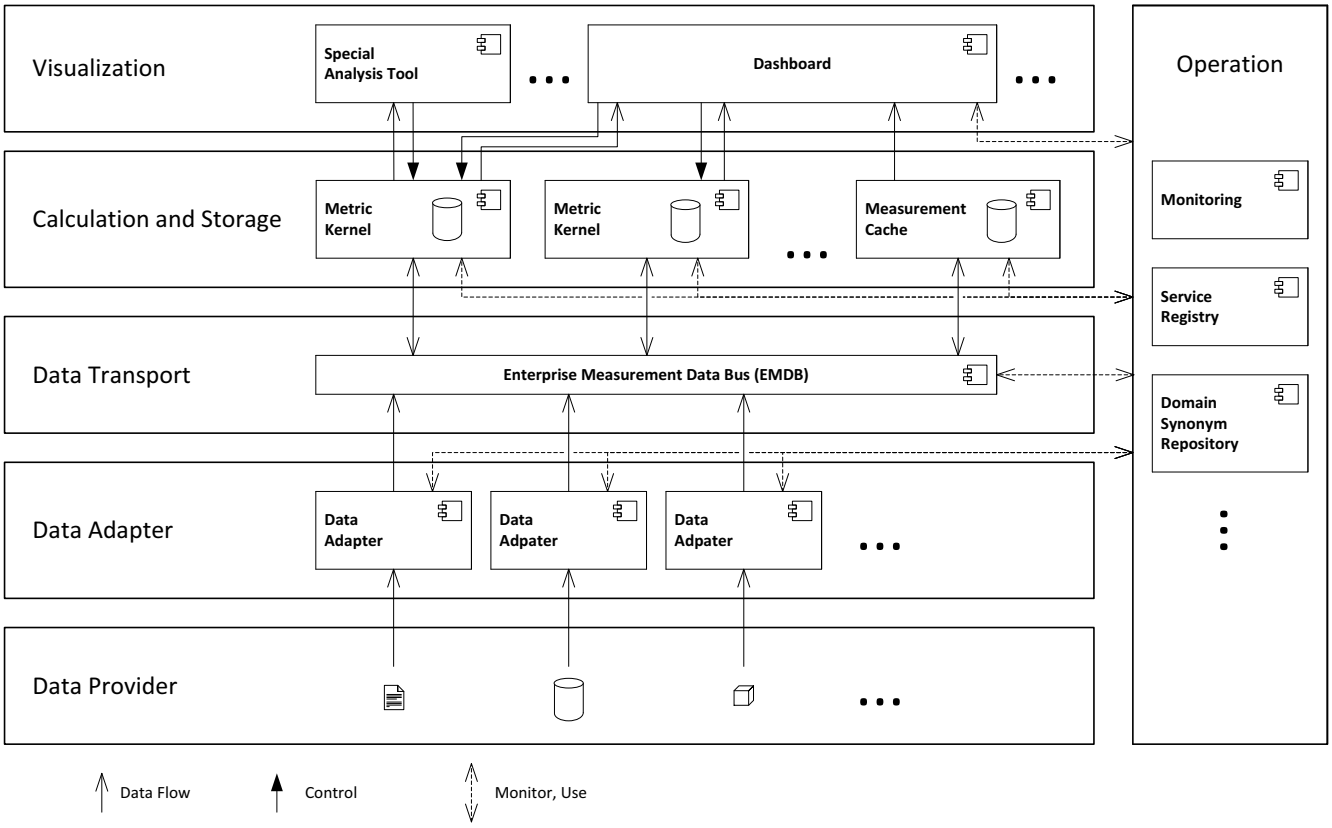

Fig. 1. Enterprise Measurement Infrastructure (EMI) components, layers, data flow, control relations, and monitor/use relations

answers which violates R2. Additionally, this still requires a central data schema in the query processor which violates requirement R6.

D. Agent based integration

Some modern integration approaches use agents to communicate between different applications [18], [19]. Agents where first used in artificial intelligence systems [20]. An agent acts in a certain environment, uses sensors to get information about it, and can use this information for its decisions. Wille and Dumke et al. propose agent based measurement tools [6], [21]. Even though agent based systems satisfy a large subset of the requirements they violate some of them. For example they are not compatible with SOAs. Additionally, they cannot provide central monitoring functionalities on their own because they are loosely coupled and are often hard to integrate into existing or new systems. Hence, agent based systems violate requirements R7, R8, and R9.

V. Enterprise Measurement Infrastructure (EMI) - Architecture and Components

Based on the requirements and typical usage scenarios we propose a layered architecture blueprint for the architecture of an enterprise measurement infrastructure which is depicted in Fig. 1. The information needs of different measurement customers are addressed by specialized analysis or dashboard tools in the Visualization Layer. The actual data needed to calculate metrics is provided by different systems in the Data Provider Layer. These systems are connected to the infrastructure using dedicated data adapters. Often visualization tools require complex and aggregated information besides pure base values. This information is produced and provided by specialized components in the Calculation and Storage Layer. The Data Transport Layer realizes a common communication infrastructure for all components of the Calculation and Storage Layer and of the Data Provider Layer. The Operations Layer contains components required to operate and monitor the complete infrastructure. In the following we explain the core concepts of the EMI architecture.

A. Dataflow

The EMI data flow depicted in Fig. 2 is based on the ISO 15939 data flow. There measurement data always flows from base measures to derived measures which are then combined in an analysis model to form an indicator that answers a particular information need. In the EMI we added important extensions, since even base measures (provided by data adapters) can form indicators (e.g. data from a tool like Sonar). Most importantly, derived measures can not only use base measures but the results of other derived measures as well as a combination of the two. *ErrorDensity* as defined in (1) is a typical example:

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

$$ErrorDensity = \frac{NumberOfErrors * 1.000}{OverallLinesOfCode} \quad (1)$$

To calculate this metric the respective metric kernel needs the current *NumberOfErrors* and the *OverallLinesOfCode*. Often, lines of code are provided by a code analysis tool like Sonar. However, these systems typically only count lines of code on a component or build fragment level. To calculate the overall *ErrorDensity* another metric kernel needs to sum up all the lines of code values from all the build fragments or components to provide a new derived measure *OverallLinesOfCode*. The *NumberOfErrors* could be calculated by a CRM metric kernel. This example shows the need of a circular data flow between different metric kernels.

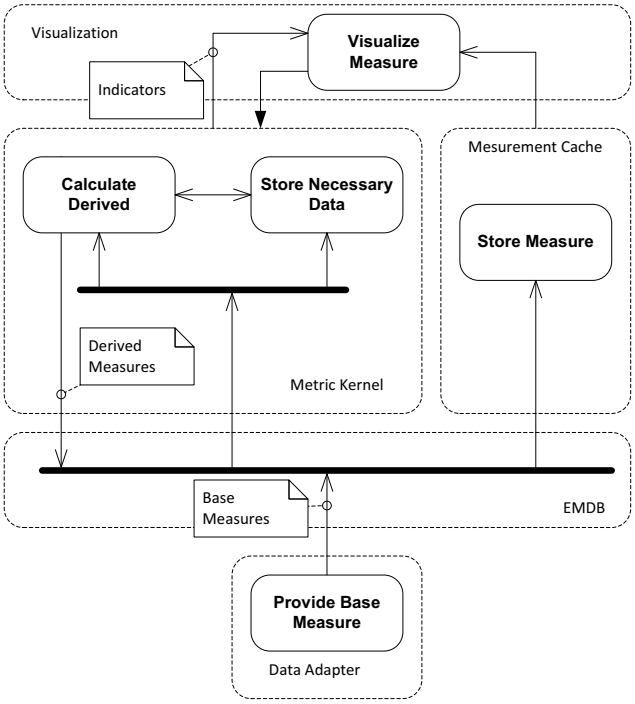

Fig. 2. Measurement and data flow in the EMI

The Measurement Cache located in the Calculation and Storage Layer is a central infrastructure component. It stores all measurement values so they are immediately accessible for visualization components. This also allows the visualization components to directly access base measures if needed. However, the tradeoff of this architectural decision is that the visualization components have to use the stored values.

B. Data Transport: Enterprise Metric Data Bus

The Enterprise Measurement Data Bus (EMDB), an implementation of an Enterprise Service Bus ([15], [22]), needs to transport the measurement values. Either from a Data Adapter (Base Measure) or from a Metric Kernel (Derived Measure) to all the Metric Kernels and the Measurement Cache of the system.

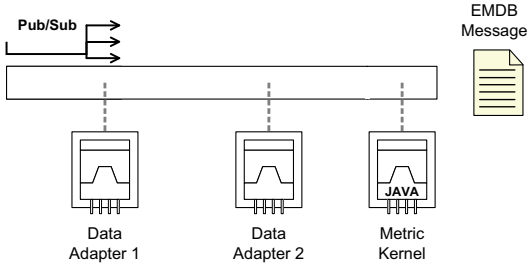

Fig. 3. Concept of the EMDB (Notation by Chappell [5])

The main concept of the EMDB, a publish/subscribe channel, is depicted in Fig. 3. It also shows two Data Adapters (as generic endpoints) and a Metric Kernel (as a Java API client). The messages that are broadcasted over the channel are of type EMDB Message (or a subtype of this). The next section describes these messages types.

C. EMDB Measurement Messages

The main design principles for the EMI are separation of concern and loose coupling. Hence, metric kernels and data adapters need to be completely separated. A metric kernel just needs to know what measures it requires for its calculations. The data provider just provides specific measures (values) for specific entities (Entities of Measurement – EOMs). Consequently the messages send over the EMDB need to inherit from a general EMDB Message type (see Fig. 4) which just defines three important attributes:

metricRefId represents the identifier (name) of the measure. We propose using a name space schema for the identifiers like

{globalNamespace}.{msgClass}. {msgSubClass}.{metric}.*

An example would be *emi.crm.NumberOfErrors* or *emi.ev.ev* or *emi.ev.pv* as well as *emi.ev.cv* for the Earned Value Analyis metrics earned value, planed value, and cost variance. This identifier is used by the metric kernels to filter the EMDB messages according to their measurement requirements.

eomId is the identifier of the EOM. It is used to provide a brought variety of measures for the same entity. The data providers typically use an internal eomId from the base systems which they adapt. The metric kernels typically reuse the eomIds from the base measures. The domain synonym repository in the operations layer can be used to build groups of eomIds. This is necessary if different systems which are adapted to the EMI use different identifiers for the same business entity.

value represents the actual measurement value. It is designed as a string to allow a brought variety of values to be transported over the bus instead of just numerical values.

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

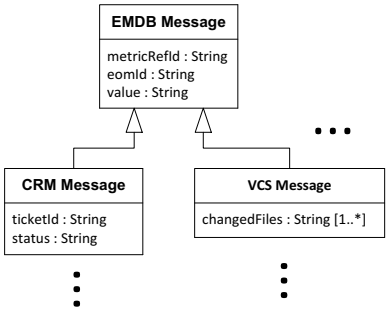

Fig. 4. Base message type hierarchy

Fig. 4 depicts the general EMDB Message with two specialized messages (CRM Message and VCS Message). The general EMDB Message can be extended by every data adapter or metric kernel that is connected to the EMDB to form specific messages that include additional information required by specific metric kernels. In general the data provider should include as much additional information with the message as possible to give the metric kernels as much additional information (for example for filtering) as possible. The CRM Message for example requires additional *ticketId* and *status* attributes. This information is useful for specialized metric kernels like our RIFFLE[3] Kernel which analyses and provides ticket flows from CRM systems.

D. Data Provision Mechanismns

The heterogeneity of the systems that are integrated into the infrastructure calls for flexible data provision mechanisms. We investigated three core provision concepts: Push-Forward, Pull-Forward, and Invoke-Push. We describe the main ideas and possible application scenarios in the following subsections.

1) Push-Forward

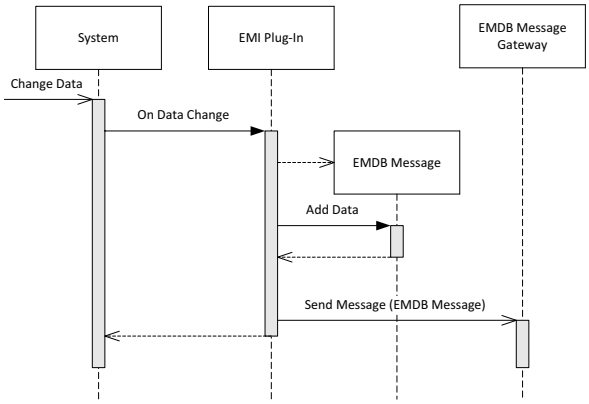

Fig. 5. Concept of the Push-Forward data provision mechanism

The Push-Forward data provision mechanism guarantees the best latency between change event in the adapted system and the visualization. The sequence diagram in Fig. 5 shows the flow of interactions. Because a plug-in mechanism in the adapted system is needed, a custom build EMI plug-in is then able to hook onto the desired change events in the adapted system. The system calls the plug-in on every data change event. Then, the plug-in creates a (specialized) EMDB message and adds specific data to the message. The message is send to the EMDB using a standard JMS Message Gateway. The data is then transported to the metric kernels and the measurement cache. Hence, the visualization components could immediately update the visualizations to reflect the new data.

2) Pull-Forward

Standard BI (Business Intelligence) systems use scheduled jobs (called ETL – Extract Transform Load) to derive data from adapted systems. The Pull-Forward data provision mechanism is inspired by these ETL jobs. Fig. 6 shows the sequence of messages. The needed EMI Extract Tasks are triggered by a scheduler who is configured to a certain interval like every minute, hour, or day. The tasks then retrieve the changed data from the systems. It should then extract the unique data chunks from the retrieved data and create a message for every chunk which is then send like push forward.

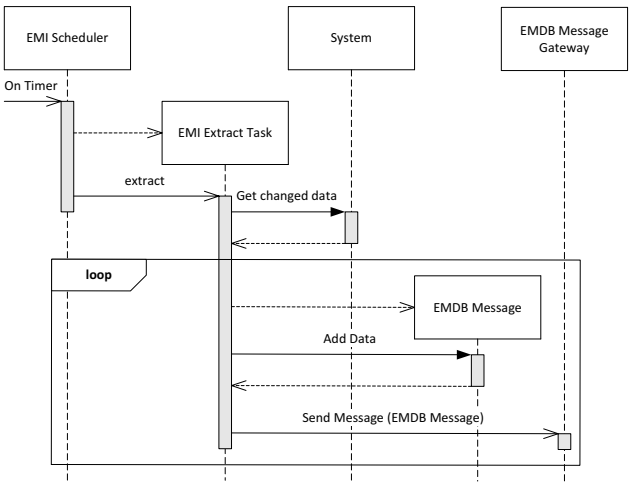

Fig. 6. Concept of the Pull-Forward data provision mechanism

Even though this provision mechanism is inspired by the most popular mechanism – ETL – it has some strong weaknesses. The most important one is latency which increases dramatically. As a result the data in the visualization is only as up to date as the latest pull interval. One solution would be to reduce the pull intervals to a minimum. However, pulling data from a system typically generates a high load in the system. Therefore, shortening the intervals will lead to performance degeneration in the adapted systems. Another weakness of this solution is the increased effort to implement the data providers.

[3] The RIFFLE Metric Kernel can use this information to identify unique tickets and provide status flows for the RIVER visualization tool to allow a detailed analysis of flows in CRM systems.

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

3) Invoke-Push

The data stored in the adapted systems typically relate to each other. For example a good practice in software development is to tag a commit into a version control system (VCS) with the task number of a task in a change request management system (CRM). The number of changed files per task could be used as a complexity measure for the task. Additionally, the number of changed lines of code could be used to normalize the effort for a task. Of course, every commit alters the number of changed files for a task. Hence, after every commit a special data adapter needs to send a new message to the EMDB containing additional information to the task. This then allows a special metric kernel to calculate the two measures.

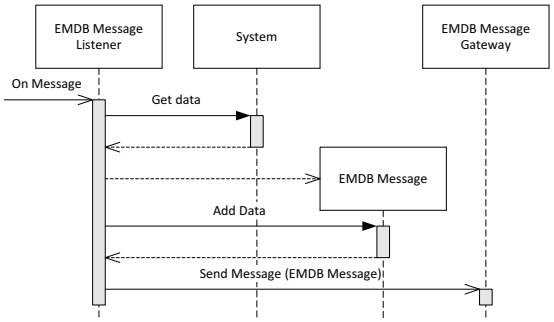

Fig. 7. Concept of the Invoke-Push data provision mechanism

Fig. 7 shows the sequence diagram of the Invoke-Push data provision mechanism which enables EMI developers to implement a special data adapter for the described situation. A special EMDB Message Listener is invoked whenever it receives a certain type of message. It then pulls data from the adapted system (for example the task from the CRM). The data is then packed into a new (specialized) EMDB message and pushed to the EMDB. This mechanism also enables a combination of Push-Forward and Pull-Forward. For example, a VCS message could be used to pull data from a changed spreadsheet file in the VCS.

E. Communication between Metric Kernels and Visualization

The measurement customer typically would like to alter some details in the metric calculation to answer more detailed or slightly tailored questions. For example the question "Are we able to address all bugs?" could be answered by the number of open bugs in a CRM system. If the project is closing in to a release date this question is typically slightly tailored to the question "Are we able to address all important bugs?" which is answered by the number of open bugs in the top categories (priority one and two).

A dashboard should allow a tailoring for these specific situations. The change in the measurement needs to be reflected by the metric kernel. This could either provide both of these metrics or the visualization component could talk directly with the metric kernel and alter the calculation of the specific metric which is feeding a certain diagram. There exist good arguments for both solutions. Hence, the EMI should allow both solutions.

Two metrics could be easily implemented in a specific metric kernel and could then feed the results back to the EMDB to allow a dashboard to access the values via the measurement cache. This solution is very elegant because it only requires the dashboard to fetch the data from the measurement cache. However, it generates additional effort in the implementation of the metric kernel because this needs to generate more derived measures. Additionally, it can lead to an explosion in the number of metrics which are communicated over the EMDB which could lead to difficulties in the maintenance and operation of the EMI. Also, this makes the measurement cache a central part in the EMI which contradicts the idea of a federalist infrastructure.

The direct communication from a dashboard to a metric kernel requires additional communication flows in the EMI (the control arrows in Fig. 1). This also increases the complexity in the configuration of the dashboard because it now needs to take the (service) source of a metric into account. However, these problems can be solved by a good and flexible framework for the communication between the metric kernels and the visualization components. We propose a solution in which the metric kernels and the dashboard can exchange instances of variability models for each metrics. These variability models include the variability points and variants for each metric. The measurement customer can then change these variability points and tailor the metrics to her specific needs.

VI. EMI Prototype and Evaluation

First prototypes for EMI components and frameworks were developed in several thesis as part of multiple industry cooperation projects [23], [24], [25]. Most importantly the dashboard tool SCREEN and several data adapters[4] and metric kernels are based on the EMI. SCREEN was successfully deployed and integrated into the software development processes and infrastructures at small and medium sized companies (the results of these field studies are published separately). We are currently planning to integrate it into larger companies with more than 250 employees (one with over 1.200). Also, we are currently starting to integrate SCREEN (and the EMI) into the software development infrastructure used by over 700 research projects at RWTH Aachen University.

Our change request analysis metric kernel RIFFLE and the visualization tool RIVER are also based on the EMI. These tools proved to be very useful to analyze CRM data (details about the tools and analysis will be published separately). Additionally, they helped to research the performance of the complete EMI. Our simulations show that a JMS based EMDB implementation and EJB/JPA based metric kernels are able to operate with over 1.500 (CRM) mes-

4 Until April 2013 we developed Data Adapters for: TRAC, Redmine, JiRA, git, svn, Excel, ClearQuest CSV dumps, Hudson, Jenkins, SONAR, generic REST, generic SOAP

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20

sages per second and an average of 1.000 messages per second on a standard notebook running a glassfish application server with OpenMQ. This allows the tools to import[5] ClearQuest CSV dumps with over 28.500 tickets in under 25 seconds which is great for development. Our predictions are that a productive environment with dedicated message bus server(s) can dramatically increase these numbers. Hence, we do not think that the EMDB will become a performance bottleneck like feared by some of our industry partners.

We are currently working on the operation components and on the framework for the variability exchange between metric kernels and the visualizations components. We are also working on several (generic) metric kernels and on several additional data adapters. All the work on the EMI implementation, metric kernels, data adapters, and visualization components in the last year showed the strengths of the infrastructure. The strong separation of concerns due to the federalist design helped to streamline the development in several simultaneous projects.

VII. Conclusion

In this paper we proposed an Enterprise Measurement Infrastructure (EMI) which is based on best practices of service oriented architectures. The EMI is based on a set of federalist systems rather than on a centralistic system to measure, analyze and visualize different data. This design decision proved to work really well in different implementation scenarios. In addition, the different parts of the EMI are well aligned with the business needs of measurement customers like proposed in the application scenario in part II.

The most important (measurement) parts of the EMI are the data flow and the data provision mechanisms. The flexible data flow together with separated metric kernels helped to implement different EMI prototypes for our field studies in parallel. We strongly belief that we are now able to integrate all the different solutions into a large toolbox that helps to address upcoming integration problems in new field studies.

Even though the intermediate results until now are very promising we still need to prove that the EMI is as maintainable and flexible as desired. Unfortunately, to answer this question we need to have EMI installations running in business contexts over a long period of time. Luckily, we already have some installations running and we are currently planning larger installations. This will help us to get valid results about the maintainability and performance of the EMI.

References

[1] C. P. Team, “CMMI® for Development, Version 1.3 CMMI-DEV, V1.3,” 2010.

[2] S. Few, *Information Dashboard Design: The Effective Visual Communication of Data*. O’Reilly Media, Inc., 2006.

[3] M. Kunz, A. Schmietendorf, R. R. Dumke, and C. Wille, “Towards a service-oriented measurement infrastructure,” in *Proc. of the 3rd Software Measurement European Forum (SMEF)*, 2006, pp. 197–207.

[4] S. Architecture, “Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus,” no. April, pp. 1–8, 2006.

[5] D. A. Chappell, *Enterprise service bus*, 1st ed. O’Reilly Media, Inc., 2004.

[6] R. R. Dumke, “Software-Messung und -Bewertung - Eine Bilanz.” 2012.

[7] K. Umapathy, S. Purao, and R. R. Barton, “Designing enterprise integration solutions: effectively,” *European Journal of Information Systems*, vol. 17, no. 5, pp. 518–527, 2008.

[8] S. Aier and R. Winter, “Fundamental Patterns for Enterprise Integration Services,” *International Journal of Service Science Management Engineering and Technology IJSSMET*, vol. 1, no. 1, pp. 33–49, 2010.

[9] T. Puschmann and R. Alt, “Enterprise Application Integration - The Case of the Robert Bosch Group,” vol. 00, no. c, pp. 1–10, 2001.

[10] R. Dąbrowski, K. Stencel, and G. Timoszuk, “Software is a directed multigraph,” *Software Architecture*, pp. 360–369, 2011.

[11] H. Wache and T. Voegele, “Ontology-based integration of information-a survey of existing approaches,” *IJCAI--01 Workshop: Ontologies and Information Sharing*, pp. 108–117, 2001.

[12] P. a. Bernstein and E. Rahm, “A survey of approaches to automatic schema matching,” *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, Dec. 2001.

[13] S. Chaudhuri and U. Dayal, “An overview of data warehousing and OLAP technology,” *ACM Sigmod record*, no. March 1997, 1997.

[14] M. P. P. and D. Georgakopoulos, M. P. Papazoglou, and D. Georgakopoulos, “Service-Oriented Computing,” *Communications of the ACM*, vol. 46, no. 10, pp. 24–28, 2003.

[15] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-Oriented Computing: a Research Roadmap,” *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, Jun. 2008.

[16] M. Kunz, A. Schmietendorf, R. R. Dumke, and C. Wille, “Towards a service-oriented measurement infrastructure,” pp. 197–207.

[17] A. Halevy, N. Ashish, and D. Bitton, “Enterprise information integration: successes, challenges and controversies,” in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 778–787.

[18] R. Kishore, H. Zhang, and R. Ramesh, “Enterprise integration using the agent paradigm: foundations of multi-agent-based integrative business information systems,” *Decision Support Systems*, vol. 42, no. 1, pp. 48–78, Oct. 2006.

[19] M. Wooldridge, “Intelligent Agents: The Key Concepts,” in *Proceedings of the 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 on Multi-Agent-Systems and Applications II-Selected Revised Papers*, 2002, pp. 3–43.

[20] S. J. Russell, P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*, Third Edit. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2010.

[21] R. R. Dumke, R. Koeppe, and C. Wille, *Software Agent Measurement and Self-Measuring Agent-Based Systems*. 2000, pp. 1–44.

[22] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, “The Enterprise Service Bus: Making service-oriented architecture real,” *IBM Systems Journal*, vol. 44, no. 4, pp. 781–797, 2005.

[23] A. Steffens, “Entwurf eines Architekturmodells zur Integration heterogener Systeme in MeDIC,” 2013.

[24] F. Evers, “Konzeptionelle Erweiterung von Projektdashboards für unerfahrene Anwender,” 2012.

[25] C. Hans, “Einsatz von Metrik-Dashboards im industriellen Umfeld,” RWTH Aachen University, 2012.

5 Importing the tickets includes reading the CSV file, parsing the data, generating messages for every ticket, sending the messages over the EMDB, receiving the messages in RIFFLE, interpreting the messages, and storing the data in the internal database of RIFFLE.

Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on, pp.63,70, 23-26 Oct. 2013
doi: 10.1109/IWSM-Mensura.2013.20