Tool Integration Beyond Wasserman

Fredrik Asplund, Matthias Biehl, Jad El-Khoury, Martin Törngren

KTH, Brinellvägen 83, 10044 Stockholm, Sweden
{fasplund, biehl, jad, martint}@kth.se

Abstract. The typical development environment today consists of many specialized development tools, which are partially integrated, forming a complex tool landscape with partial integration. Traditional approaches for reasoning about tool integration are insufficient to measure the degree of integration and integration optimality in today's complex tool landscape. This paper presents a reference model that introduces dependencies between, and metrics for, integration aspects to overcome this problem. This model is used to conceive a method for reasoning about tool integration and identify improvements in an industrial case study. Based on this we are able to conclude that our reference model does not detract value from the principles that it is based on, instead it highlights improvements that were not well visible earlier. We conclude the paper by discussing open issues for our reference model, namely if it is suitable to use during the creation of new systems, if the used integration aspects can be subdivided further to support the analysis of secondary issues related to integration, difficulties related to the state dependency between the data and process aspects within the context of developing embedded systems and the analysis of non-functional requirements to support tool integration.

Keywords. Tool Integration, Model-based Tool Integration, Model-based Development, Integrated Development Environments

1. Introduction

We work within the domain of embedded systems. In this domain there is traditionally a strong emphasis on ensuring system quality throughout the development process. This is due to embedded systems, through their close interaction with the environment in domains such as transportation and automation, have a strong presence of requirements on safety, availability, performance, etc. To ensure relevant expert knowledge development efforts are therefore often very much multidisciplinary, which leads to many stakeholders and a large diversity of tools. In development environments for embedded systems well functioning tool integration is needed to achieve seamless cooperation between tools to support the goals of the tool users. Due to it being unlikely that the whole system will be taken into account during every change done to improve tool integration this strive towards seamless cooperation typically leads to *islands of automation* (i.e. integrated groupings of tools). These islands support only specific parts of the *product life-cycle* and are heterogeneous environments that can vary drastically in the way they carry out tool integration.

We have used the principles that originated in [1] to build a reference model for reasoning about tool integration in these heterogeneous development environments (hereon after this is the *context* of which we speak), since we found the original principles insufficient for our work. The second section in this paper describes the reasons behind diverging from the original principles and how we do this by adding metrics and dependencies to the categories defined in [1].

The third section describes a method on approaching tool integration in our context. This is done both to exemplify our reference model and to allow for it being used in case studies.

In the fourth section we describe a case study.

In the fifth, and final, section we sum up our findings and point out the research directions in the greatest need of further effort.

2. Diverging from Wasserman

To facilitate the reasoning about tool integration in an existing system we need:

- A set of *categories* that divide an integration effort into parts that can be discussed in isolation. This is needed to structure the reasoning into manageable parts.
- A way to *measure* the degree of integration. This to needed to understand the state of the integration of a system.
- A notion of what *optimal* integration is. This is needed to know in which direction to move when improving a system.

[1] provides all of these, but before we settled on this set of principles we performed an extensive literature review to judge its current value. As will be apparent later in this section these principles figure prominently in literature throughout the last two decades, which is, in itself, promising. What made us finally decide was that [1] is being mentioned in literature reviews summarizing the most salient papers [2] and used as a reference to compare with when trying to build new research agendas for tool integration [3].

These principles are, however, not fully applicable in our context. In the following subsections we will discuss each of the requirements above as put forward in [1] and how we diverged from them when building our own reference model.

2.1. A Set of Categories

First, there are five widely accepted categories found in [1], namely the *control, data, platform, presentation* and *process integration* aspects (hereon after *aspects* is used to specifically indicate these five categories). For clarity we have listed the definitions of these aspects below. These definitions are in essence the same as the ones found in [1], with only minor updates to reflect our view of the discussion since the paper was published.

- Control integration is the degree to which tools can issue commands and notifications correctly to one another.

- Data integration is the degree to which tools can, in a meaningful way, access the complete data set available within the system of interest.
- Platform integration is the degree to which tools share a common environment.
- Presentation integration is the degree to which tools, in different contexts, can set up user interaction so that the tool input/and output is perceived correctly.
- Process integration is the degree to which interaction with tools can be supervised.

This decomposition is often used when structuring development efforts or case studies. Discussions often lead to which *technical mechanisms* (such as explicit message passing vs. message servers, etc., hereon after mentioned only as *mechanisms*) should be covered ([4]) or were used to address which integration aspect ([5], [6], [7], [8], [9], [10], [11]). These discussions often take a "broad" view of what the different aspects entail and discuss only the mechanisms actually used.

The aspects are used also in evaluative approaches. In [12] the aspects are used for categorizing, but combined with the notion of integration levels [13] to define the extent of the integration. An attempt is made to relate the different levels to several of the aspects. In the end this is again done by directly mapping different mechanisms to the different levels. In [14] all the aspects are pulled into one category where different groupings of mechanisms determine the level of integration.

So, while the aspects are widely used, they are used as vague categories. When exactness is needed the mechanisms are the preferred fallback. This is natural, since in [1] mechanisms are used as the measure of integration. Assigning different mechanisms to different aspects has with time also become the way of isolating the aspects from each other when exactness is needed.

2.2. A Way to Measure

There are two reasons why using mechanisms as the measure of integration is problematic for us. First, it does not allow discussing complex mechanisms that cover several aspects without the terminology becoming vague [15]. Secondly, this approach does not scale as the amount of mechanisms grows. Both of these problems are especially difficult in our context, since we deal with such heterogeneous environments. Therefore, to facilitate the reasoning about integration in our context, we have come up with a new *metric* for each of the aspects. These are described in the paragraphs below.

We define our metric of control integration as the number of unique services in a system. This we have picked up from the notation in [16] that the provision and use of services is the essence of this aspect. The possibility to achieve a high level of integration increases the finer the granularity of a systems services is.

We define our metric of data integration as the number of data interoperability sets in a system (a data interoperability set is a collection of syntax and semantic pairs that are related by links or transformations). This we have deduced from the discussions on data integration ([16], [17]) where data interoperability figures prominently. The possibility to achieve a high level of integration decrease when the number of data interoperability sets increase.

We define our metric of platform integration as the distinct platforms used in the system. The more platforms that have to be considered when analyzing the other aspects, the less possible it is to achieve a high level of integration.

We define our metric of presentation integration as the level to which users are forced to view or manipulate data in views they are not used to interact with. Most of the discussions about presentation integration are centered on the need for a common "look and feel" ([16], [18], [19]). However, in our context a specific user is usually not expected to use all of the tools present and thus a lot of different "look and feels" can exist without a major impact. The more data entities that are accessed in different views, pertaining to different development activities, the less possible it is to achieve a high level of integration.

We define our metric of process integration as the number of services that allow the end-user to know the state of the system when they signal completion (implicitly or explicitly). Process integration is mostly discussed in relation to frameworks put forward to facilitate the application of a process ([15], [20], [21], [22]), but we focus on process in its most simple form. The closer this metric comes to the total number of services, the better the possibility to achieve a high level of integration.

Our metrics, however, do not help us to isolate the aspects from each other. To be able to use them we also explicitly list the dependencies between the aspects.

The different services of control integration need to be reachable, which is a feature of platform integration. Data also needs to be reachable, which is a combined feature of control and platform integration. Control, data and platform integration limit the choices available for presentation integration. The logic of process integration will depend on control (process awareness) and data integration (state). These dependencies are visualized in *Figure 1*.

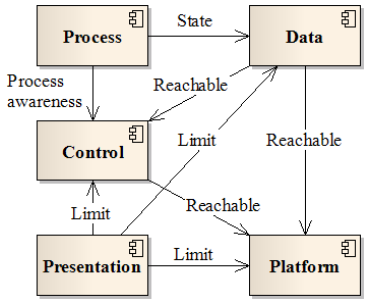

Figure 1. Dependencies between aspects

We should treat the dependencies as entities in their own right and understand how they are handled. Then the aspects can be discussed and measured in isolation.

2.3. A Notion of Optimality

This leads us to the last issue, the notion of optimality. In [1] optimal tool integration is defined as when all tools agree on the mechanisms in regard to all aspects. This is of course true, but the price for and time to achieve such a homogeneous environment is likely to be large. We need a reference model which is able to point out what to focus on and in what order. The dependencies and our metrics work together to give our reference model this property.

The first indicator of optimality is the existence of as few dependencies *as possible* (process steps may for instance dictate a certain number of required dependencies). Fewer dependencies indicate both less need for and a greater possibility for integration between aspects. The structure of dependencies provides the order in which to deal with the aspects, from the aspect with no dependencies (platform) to the ones at the end of the dependency chains (presentation and process). A change at the beginning of a dependency chain will affect more entities than a change at the end of a chain. Our metrics are the secondary indicator of optimality.

3. A Method to Analyze Development Environments

To exemplify the reference model described in the previous section and to prepare the way for an upcoming case study we here describe a method for analyzing development environments. The description will make use of a fictional example of three islands of automation, shown in *Figure 2*.

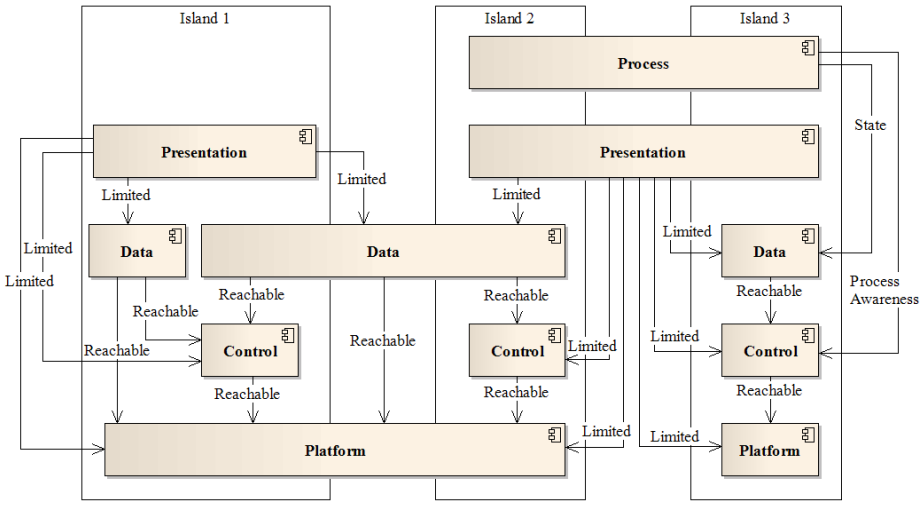

Figure 2. A fictional example of three groupings of integrated tools illustrated through their integration aspects and dependencies

3.1. Identify the Member Sets and Their Dependencies

The first step is to identify all the sets of members of our metric sets (i.e. the sets of related data interoperability sets, the platforms used together, etc.) and their dependencies. *Figure 2* illustrates this.

The implications of the first three aspects (platform, control and data) in the dependency chain are obvious, but we note that they can be shared between islands. When none of these member sets are shared this implies a data artifact which is manually transferred between development steps (using a word processor or such). Less obvious is perhaps the sharing of a presentation and process member set. For a presentation member set this simply implies that the same abstractions are supported on both islands, perhaps by adhering to a standard (UML, etc.). For a process member set this implies that there is no possibility to identify distinct process states on the different islands, perhaps because the process is one of gradual refinement. The later could be equally well signaled by not listing a process member set (note the dependencies between control, data and process however, where there is none it implies that the completion of a process step cannot be known by the tools). We also note that data member sets can be shared even though control member sets are not; this simply implies the use of different data bases or such.

3.2. Reason about Aspects

The second step is to reason about the aspects on a more general level. This step is also undertaken to identify and investigate the non-functional requirements (available expertise, cost-effectiveness, etc.) that block the strive towards a homogeneous, non-fractured environment. If it is obvious that a large part of the non-functional requirements can be changed, then we can jump directly to reasoning about them and then start over afterwards (to avoid sub-optimization). Important issues not strictly dependent on the optimality of integration, such as data synchronization, interaction paradigms in use, etc, should also be discussed during this step.

3.3. Add Member Sets and Dependencies

The third step is to review missing member sets and dependencies, to see if something can be gained by introducing extra complexity. In our example this could consist of introducing separate, identifiable process steps on island 1 and 2.

3.4. Minimize the Number of Member Sets

The fourth step is to minimize the number of member sets, which entails both reducing the amount of dependencies and maximizing our metrics. In our example the former could consist of using the same database for all data on island 1 and 2, using the same platform set for all islands, etc. The later could consist of using fewer platforms on a particular island, increasing services that signal completion, etc.

3.5. Reason about Non-functional Requirements

The last step would be to reason about which of the existing non-functional requirements can be changed. A change in these opens up the possibility to reiterate the earlier steps to identify new opportunities for better tool integration. We have not found any suggestion on a structured approach to this, which is not surprising as for instance the lack of research into economical implications of tool integration have been noted in [3].

4. A Case Study

In this section we take a look at an industrial case study put forward by an industrial partner. The case aims to assess the impact of tool integration in a development environment aimed at multicore products and featuring HW/SW co-design. The product used to evaluate the development environment is a prototypical control system for an elevator. This control system is developed to be suitable for closed loop control where a number of sensor elements and actuators are connected by various interfaces. The system performs relevant actions depending on the input signals, the internal system state, the configurable logic, and operator commands. The development is to a large degree done according to the principles of *Model-based Development* (MBD).

Table 1. Original set of tools analyzed in the case study

Req. Engineering & Analysis	Design	Implementation	V&V
MS Word	Matlab/Simulink with RTW	Windriver Workbench	Matlab/ Simulink
Excel	Control Builder (1131)	Xlinx ISE	ModelSIM
MS Sourcesafe	MS Sourcesafe	MS Sourcesafe	MS Sourcesafe
PCVS	PCVS	PCVS	PCVS
	MS Word	GNU	
	Excel		

The total number of tools that figured in the original setup was 15. Out of these we limited ourselves to 11 (see *Table 1*), disregarding tools whose use was not clearly defined. The most obvious problem was the fragmented handling of data, both in regard to the data and control aspects (see *Figure 3*). In total we identified 12 separate data interoperability sets, all potentially handled separately from each other in 17 different development steps. The lack of process support was also obvious, since no process tool or integrated process support in the tools involved were used.

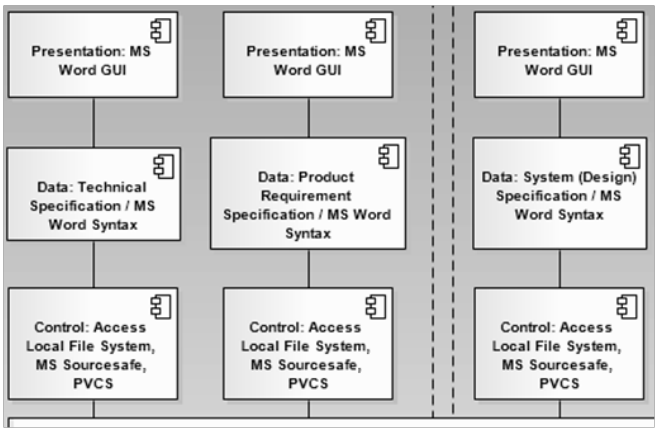

Figure 3. Part of the visualization highlighting the data fragmentation

After identifying the original state of the development environment we analyzed it according to our method introduced in section 3 and came up with suggested improvements. However, there were few non-functional requirements defined for the case study. Therefore we had to evaluate our solution by contrasting it against suggested improvements by system engineers working at our industrial partner. The commonly identified improvements we analyzed in a first iteration consisting of relatively easily achieved integration improvements. Among the additional improvements we had identified we chose two (discussed further below) which will be the focus of further research and included them in a second iteration. Due to the lack of non-functional requirements we could not evaluate the worth of the remaining improvements highlighted by our reference model and removed them from the case study.

Due to the existence of a defined product life-cycle we could exchange the use of MS Word and Excel early in the development process with the use of Enterprise Architect. This allowed the start of MBD practices earlier than before. The use of EDA Simulator Link allowed us to link Hardware and Software Development. Together this shrank the total number of data interoperability sets to 4. The introduction of SVN and a MS Windows network similarly simplified the reachability of the data.

The first of the two improvements in the second iteration was to introduce traceability between requirements and design. This would shrink the total number of data interoperability sets even more, reducing error sources and increasing the possibility of verification and validation. The second improvement related to the lack of process support. This would entail introducing support for the process awareness dependency and moving the knowledge of the state of the process from inside the heads of the designers to the data.

4.1. Evaluating our Reference Model

The critical issue is not the improvements of the development environment in the case study; it is how the use of our reference model compares to the use of [1] when searching for these improvements. To evaluate this we made use of integration weaknesses identified in the case study.

There were a total of 25 integration weaknesses identified by domain experts and developers using this tool chain on a daily basis. Note that 14 out of these 25 were not relevant for this comparison:

- 4 were domain issues such as the lack of a complete list of requirements. These were contained within the aspects in both reference models.
- 4 detailed secondary issues, i.e. workflow optimization, version control, etc.
- 1 was in direct conflict with a non-functional requirement we had identified.
- 5 were related to tools we had excluded from our analysis or were too vague to be analyzed.

7 of the remaining 11 integration weaknesses were equally visible in both reference models. Interestingly enough almost all of them related to data being handled inefficiently and in a segmented way. The remaining 4 were obvious in our reference model, but not when using the principles detailed in [1].

- While there was a development process, it was not rigorously defined. This relates to the lack of process state defined for the data, which we could identify since there was no state dependency for several of the process steps (see *Figure 4*).
- The process was customized by the different designers. This was identified like in the issue above, but due to missing process awareness dependencies.
- There was an unnecessary diversity in the tool solutions. The limitation dependency between presentation and data clearly identified the difference in tool boxes being used in the same development phase, but on different system parts (see *Figure 5*).
- Some data was not possible to access from relevant development steps, once again identified by the dependencies between presentation and data.

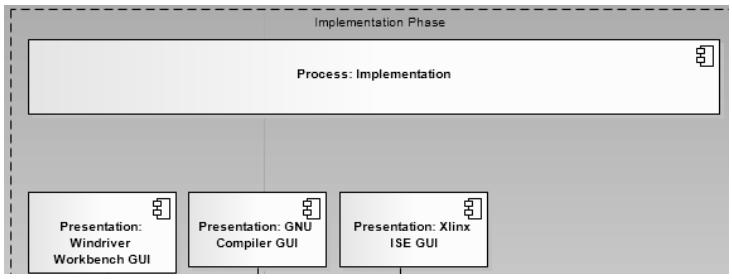

Figure 4. Process step without dependencies to control or data

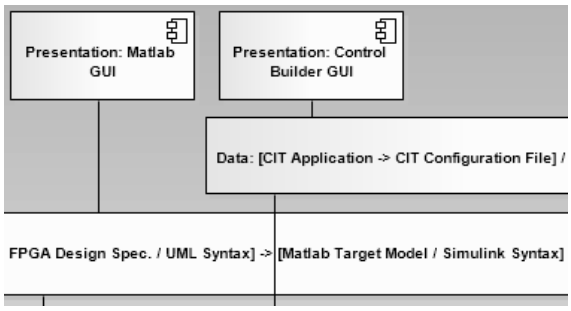

Figure 5. Tool access related to relevant data

The completeness of our classification framework can be shown for this case study, as all 20 integration weaknesses could be identified and isolated to an aspect or dependency.

5. Discussion and conclusion

We have found that we cannot apply the decomposition defined by Wasserman in [1] when reasoning about integration issues in the heterogeneous environments we work with. To solve this we have extended the set of principles in [1] into a new reference model. This model moves away from technical mechanisms as the measurement of integration. At a higher level of abstraction it instead focuses on reasoning around the dependencies between and metrics for integration aspects.

We have analyzed a limited case study to evaluate our reference model. For that case we found no sign of deterioration of the usefulness of the principles found in [1]. On the contrary, a number of integration weaknesses that would not have been obvious earlier were now highlighted. We were also able to break down and isolate all integration issues in the case study to individual aspects or dependencies.

Our ideas show promise, but there are a number of open questions. We think the most important of these are:

- Is our reference model suitable for decision support when building a new development environment? While the principles in [1] compare different mechanisms in an absolute way our reference model focuses on giving guidance on how to improve a system. To ensure the usefulness in this scenario our reference model should be applied to different architectures and platforms to identify how our metrics relate to them.
- The current metrics are very generic, while there are a number of secondary issues related to integration (data synchronization, data consistency, etc.). The later can be handled by checklists or multi-dimensional reference models, but these may limit users by only highlighting a subset of the relevant issues. Instead, some of the aspects could potentially be divided into parts if dependencies between these parts were identified. This could give guidance without limiting reasoning.

- In our experience, the identification of mechanisms to support the different aspects and dependencies is usually easy. It is however difficult for the state dependency between the data and process aspects, at least within the context of the development of embedded systems. This is due to difficulties in ascertaining the state of the data artifacts in relation to the development process. These difficulties are related to a diversity of issues, such as rules for artifact refinement, development metrics, requirements traceability, optimizing of different system aspects, etc. ForSyDe [23] is an example of a design methodology that takes this into account, but actual tool support is not easy to come by.

In addition there is one broader question that is not peculiar to our reference model, but which is critical to its use (as discussed in section 3):

- How non-functional requirements relate to tool integration is another very relevant topic which requires additional research. Are there separate aspects yet to be defined to support reasoning about these, and what are the involved dependencies in that case?

6. Acknowledgments

We thank all participants of the ARTEMIS iFEST project, who have given us continuous access to an additional breadth of expertise on and experience of software engineering in relation to the life-cycle of embedded systems.

7. Bibliography

1. Wasserman, A.L.: Tool Integration in Software Engineering Environments. In: Long F. Software Engineering Environments: International Workshop on Environments, Chinon, France, September 1989, Proceedings. pp. 137-149, Springer-Verlag, Chinon, France (1990)
2. Wicks, M.N.: Tool Integration within Software Engineering Environments: An Annotated Bibliography. HW-MACS-TR-0041. Heriot-Watt University, Edinburgh (2006)
3. Wicks, M.N., Dewar, R.G.: Controversy Corner: A new research agenda for tool integration, J. Syst. Software. 9, Vol. 80, ISSN: 0164-1212. Elsevier Science Inc., New York, NY, USA (2007)
4. Chen, M., Norman, R.J.: A Framework for Integrated CASE, IEEE Software. 2, Vol. 9, ISSN: 0740-7459. IEEE Computer Society Press, Los Alamitos, CA, USA (1992)
5. Gautier, B., et al.: Tool integration: experiences and directions. In: Proceedings of ICSE '95. ISBN: 0-89791-708-1. ACM, New York, NY, USA (1995)
6. Bao, Y., Horowitz, E.: A new approach to software tool interoperability. In: Proceedings of SAC '96. ISBN: 0-89791-820-7. ACM, New York, NY, USA (1996)
7. Best, C., Storey, M.-A., Michaud, J.: Designing a component-based framework for visualization in software engineering and knowledge engineering, In: Proceedings of SEKE '02. ISBN: 1-58113-556-4. ACM, New York, NY, USA (2001)
8. Wallace, E., Wallnau, K.C.: A situated evaluation of the Object Management Group's (OMG) Object Management Architecture (OMA), SIGPLAN Not.. 10, Vol. 31, ISSN: 0362-1340. ACM, New York, NY, USA (1996)
9. Reiss, S.P.: The Desert environment, ACM Trans. Softw. Eng. Methodol. 4, Vol. 8, ISSN: 1049-331X. ACM, New York, NY, USA (1999)

10. Mampilly, T., Ramnath, R., Irani, S.: PFAST: an eclipse-based integrated tool workbench for facilities design, eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange. ISBN: 1-59593-342-5. ACM, New York, NY, USA (2005)
11. Biehl, M., DeJiu, C., Törngren, M.: Integrating safety analysis into the model-based development toolchain of automotive embedded systems, SIGPLAN Not.. 10, Vol. 45, ISSN: 0362-1340. ACM, New York, NY, USA (2010)
12. Cuthill, B.: Making sense of software engineering environment framework standards, StandardView. 4, Vol. 2, ISSN: 1067-9936. ACM, New York, NY, USA (1994)
13. Brown, A.W., McDermid, J.A.: Learning from IPSE's mistakes. In: IEEE Software. 2, Vol. 9, ISSN: 0740-7459. IEEE Computer Society, Pittsburgh (1992)
14. Baik, J., Boehm, B., Steece, B.M.: Disaggregating and Calibrating the CASE Tool Variable in COCOMO II, In: IEEE Trans. Softw. Eng. 11, Vol. 28. ISSN: 0098-5589. IEEE Press, Piscataway, NJ, USA (2002)
15. Pohl, K., Weidenhaupt, K.: A contextual approach for process-integrated tools, SIGSOFT Softw. Eng. Notes. 6, Vol. 22. ISSN: 0163-5948. ACM, New York, NY, USA (1997)
16. Thomas, I., Nejmeh, B.A.: Definitions of Tool Integration for Environments. In: IEEE Software. March, Vol. 9, 2, pp. 29-35 (1992)
17. Holt, R.C., et al.: GXL: a graph-based standard exchange format for reengineering, Sci. Comput. Program. 2, Vol. 60, ISSN: 0167-6423. Elsevier North-Holland, Inc., Amsterdam, The Netherlands (2006)
18. Tilley, S.R.: The canonical activities of reverse engineering, Ann. Softw. Eng. 1-4, Vol. 9, ISSN: 1022-7091. J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA (2000)
19. Stoeckle, H., Grundy, J., Hosking, J.: A framework for visual notation exchange, J. Vis. Lang. Comput. 3, Vol. 16, ISSN: 1045-926X. Academic Press, Inc., Orlando, FL, USA (2005)
20. Pohl, K., et al.: PRIME---toward process-integrated modeling environments, ACM Trans. Softw. Eng. Meth. 4, Vol. 8, ISSN: 1049-331X. ACM, New York, NY, USA (1999)
21. Endig, M., Jesko, D.: Engineering Processes - On An Approach To Realize A Dynamic Process Control, J. Integr. Des. Process Sci. 2, Vol. 5. ISSN: 1092-0617. IOS Press, Amsterdam, The Netherlands (2001)
22. Sharon, D., Bell, R.: Tools that bind: creating integrated environments. In: IEEE Software. March, Vol. 12, 2 (1995)
23. Sander, I., Jantsch, A.: System modeling and transformational design refinement in ForSyDe. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. January, Vol. 23, 1 (2004)