Information Processing Tools and Environments

Guntis Arnicans

Faculty of Physics and Mathematics
University of Latvia
Raina Blvd. 19, Riga LV-1586, Latvia
garnican@lanet.lv

Abstract

The various ways exist how we can build an information system. We offer to look at a information system like a suit of tools that are integrated into a collaborative environment. The other tools are used to design, implement and test such environment. They increase a convenience, productivity and quality of development process providing the implementation of target tools, its integration and satisfying the development methodology and requirements. These supportive tools make a specific software environment. In the paper we present basic things what the developers have to know following to this principle to build an information system: 1) the concept of a tool, 2) the classical approaches for tools' integration, 3) the principles of tools development, management and control, 4) the questions before tool designing, and 5) the possible conceptual architecture of a tool. The mentioned things are equally referred to both the information processing tools and the development supporting tools.

1 Introduction

Data is a formal representation of facts or ideas with possibility to be communicated or manipulated by some automated process. Information is a meaning that humans assign to data during automated data processing using the definite habits to present it (meaning of data). Disinformation is information with delusive meaning and/or off-grade data was being used to produce information.

It is increasingly difficult to draw a line around an application system and say that you own and control it. Data is distributed over a multitude of heterogeneous, often autonomous information systems, and an exchange of data among them is not easy. Let us look at a relatively simple situation – the data source and services are located in one organization but the information consumer (user) is located into another organization (Figure 1).

Process how the data is transformed to information and presented is long and difficult (going through many applications, operating systems, defense systems, data transmission protocols, etc.). If the system builders make mistakes in some part of this process, then we receive a disinformation but not the desirable information. The

problem become more serious when we produce information from many data sources, and when the services must work without interruptions.

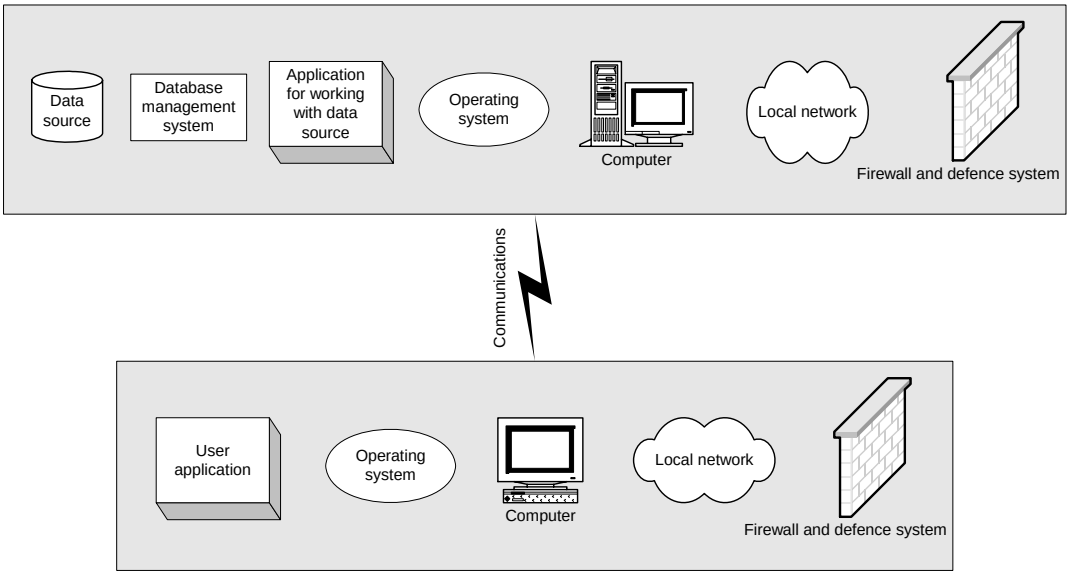

Figure 1 The information processing in heterogeneous environment

These problems emphasize the need for tools to mediate between databases, servers and front-end application. And we need tools to create these mediator tools also. We need to maintain descriptions of data structures, content, data properties, available services (metadata of data sources and services). It increases the need for dynamic manipulating of both data and metadata. Besides we have to worry about system quality that leads to need for testing, simulating and monitoring tools.

Over time, the number and variety of tools has grown tremendously. They range from traditional tools like editors, compilers and debuggers, to tools that aid in requirements gathering, design, building GUIs, generating queries, defining messages, architecting systems and connecting components, testing, version control and configuration management, administering databases, reengineering, reverse engineering, analysis, program visualization, and metrics gathering, to full-scale, process-centered software engineering environments that cover the entire lifecycle, or at least significant portions of it [HOT00].

2 The Concepts of Information Processing Tools and Environments

Any system that assists the programmer with some aspect of programming can be considered a *programming tool*. Similarly, a system that assists in some phase of the software development process can be considered a *software tool*. A *programming environment* is a suite of programming tools designed to simplify programming and thereby enhance programmer productivity. A *software engineering environment* extends this to software tools and the whole software development process [Rei96].

The definitions above we can refer to any software. Let us look to the narrower class of software – a software for information processing. Similarly to previous definitions we introduce the concepts of information processing tool and information processing environment.

An *information processing tool* is any system that provides performing some task while information processing. An *information processing environment* is a suit of information processing tools that together makes intended information processing. On the other hand, from the perspective of end-user the information processing environment is simply an *information system*.

An *information system software tool* (simply a *tool* below in the text) is a system that assists in some phase of the information system development process. And finally, an *information system software environment* (simply an *environment* below in the text) is the extension of the information processing environment with the information system software tools.

3 Classification of Information Processing Tools

Tools can be categorized by the phase of information system development and the particular problem that they solve. It is possible categorize them also by development principles, integration principles, runtime behavior, etc.

3.1 Classification by the Functionality

One of the most natural ways to classify the tools is grouping them by its functionality. Many grouping principles exist. We offer to classify them in the following way (only an example how it can be done):

1. Data extracting tools from data sources

    1.1. Relation databases
        1.1.1. Tool for the specific database
        1.1.2. Universal tool for various databases
    1.2. Object-oriented databases
    1.3. Structured files
    1.4. Other data source
2. Communication tools to work with data sources
    2.1. Specific client software for particular data source
    2.2. Internet Browser
    2.3. Email
    2.4. Other communication tool
3. Tools for describing of data sources
    3.1. Repository for data sources descriptions
    3.2. Data source describer
    3.3. Data source services describer
    3.4. Other
4. Inquiry processing tools
    4.1. Inquiry definition tools
    4.2. Inquiry executing tools
    4.3. Other
5. User interface generating tools
    5.1. Static, predefined interface application generator
    5.2. Dynamic, varying interface application generator
    5.3. Other
6. User management tools
    6.1. User registration and general management tools
    6.2. User rights management tools
    6.3. Finances accounting tools for services
    6.4. User profiles management tools
    6.5. Other
7. System auditing tools
    7.1. Audit journals management tools
    7.2. Statistics accounting tools
    7.3. Security control tools
    7.4. Other
8. System quality tools
    8.1. System testing tools
    8.2. Documentation tools
    8.3. Other
9. User temporal data management tools (temporal databases)
10. Other

3.2 Classification by the Runtime Behaviour

The other principle to classify the tools is grouping them by its runtime behavior. Many grouping principles exist. We most of tools divide into two basic groups:

1. Static tool. The tool performs the specific predefined and fixed functionality, and this functionality can be changed only by redesigning and implementing of the

tool. It is possible that functionality can be altered by some simple predefined configuration functions or by configuring these tools before running them. Usually input data is in relative strong predefined format. Basically the source code compilation is used to obtain executable software (tool).

2. Dynamic tool. The tool can vary its own functionality or in the other words – change executing semantics before or during the tool operating time. It is possible to vary functionality in large range. Input data format and meaning can be different. The interpreter is more preferable to implement such tools. The tool works interpreting commands received as a program before or during the operating.

4 Tools Integration

We mentioned above that from the user perspective information processing environment is an information system. If we build the information system as a set of tools, then we need to integrate all tools into a collaborative work. These tools can be combined together in a variety of ways using various integration techniques.

System developers choose integration techniques being guided by practical needs, system complexity, knowledge and skills of developers, etc. It is distinguished three the most popular approaches for an integration that involve ways for the tools to share information and interfaces [Rei]:

1. Data integration. It assumes that tools share information. Usually a database or repository is created. Most of the tools stores and consumes shared information. Via this repository all tools work with the same data and data exchanging also is organized through the repository.

2. Common front end. The user sees and uses all system together. He does not exploit any tool separately and often does not know that the system consists from a set of tools. The user operates with data objects and operations allowed at the specified moment. Usually the common interface is used to integrate tools, and tools do not share information.

3. Control integration. This approach involves message passing between the tools. Tools send messages to other tools whenever they need to share information or whenever a command from one tool is invocated from another.

The message exchanging mostly is organized through a central message server. The tool send a message to the server, and the server send this message to all other tools that are interested in to this message type. The tools can exchange with messages directly, but this approach can arise problems if the amount of tools is large.

A combination of the all integration types is used to develop serious and large environment of integrated tools.

5 The Principles for the Tools' Development, Management and Controlling

Real world changes all the time, and requirements to systems also changes. We have to take into account these changes and to implement them into our information system. In the distributed and heterogeneous computing environment it is a serious problem. Besides the nowadays' services must run without interruptions, and system changes must be done by changing behavior of the tools sending them a new configuration or replacing them dynamically with new tools. The tools have to satisfy the following requirements – convenient configuration and control facilities, a possibility to change behavior semantics, an acceptance of various input and output formats, etc.

To deal with these problems and to provide convenient means for tools integration and running system maintenance we advice exploit common principles of the tools' development, management and controlling. The most important principles are:

1. Common architecture. Most of the tools have a similar architecture. This allows a designing of the tools with common components. The tool development and maintenance becomes less resources consuming, and quality of resulting tool is higher.
2. Common software (modules). If the ideology and architecture of the tools is similar, then an implementation can contain common modules, subsystems, runtime libraries, etc.
3. Common configuration mechanism. It assumes that we can change the behavior of tool (predefined changes without software changes) dynamically

in similar way according to the specific task. These leads to common modules and easier exploiting, integration and maintenance of tools.

4. Common control mechanism. The tools have to have standardized means (interfaces) to control and manage them. Via this interface user or other tools gives commands to the specific tool, and it is a main mechanism for the tools integration.
5. Common monitoring mechanism. Information systems usually have to work in an uninterrupted regime. We need to monitor system operating, find the weakest points and perform some actions to correct the system performance.
6. Common testing principles and means. The system quality is crucial topic for systems. We have to test tools before deployment, and common principles and testing tools can reduce most expensive resource costs (time, money, people). Moreover, we have to continue testing while real system exploiting and check every operation if we use dynamic code generation and immediate just-in-time compilation or interpretation.

6 The Conceptual Architecture of a Tool

There are many various opinions what the conceptual architecture of a tool looks like. Before we present our model let us state the most important questions to understand the essence of a tool:

- What is the main task for desired tool? Why does it necessary us?
- How can we build this tool? What are the possible technologies, data structures and algorithms? Can we use or adapt existing tools or modules?
- What is an input for our tool? Does a tool receive all input data before starting computations or get it by portions?
- What is an output for our tool? Does a tool produce all output data after computations or supply it by portions?
- Is a tool stateless or not? Does a tool remember the history about previous computations?

- What is a way to manage and control the tool? What is a desirable interface for such tool?
- Do we need to change the behavior of a tool dynamically without interrupting all running environment or moreover while operating time? What are the things we need to change (configuration, executing semantics, input/output formats, etc.)?
- Have we got necessity to monitor the state and behavior of a tool before, during or after operating time?
- Does a tool cooperate with other applications or tools excluding desired "official input and output"? Is this cooperation synchronous or asynchronous?
- Can our tool change the state/configuration of other application or tool? Can the other tool change our tool state/configuration? At what time (before or during operating)?

We have created a conceptual model of tool architecture taking into account questions above. The model is shown in Figure 2.

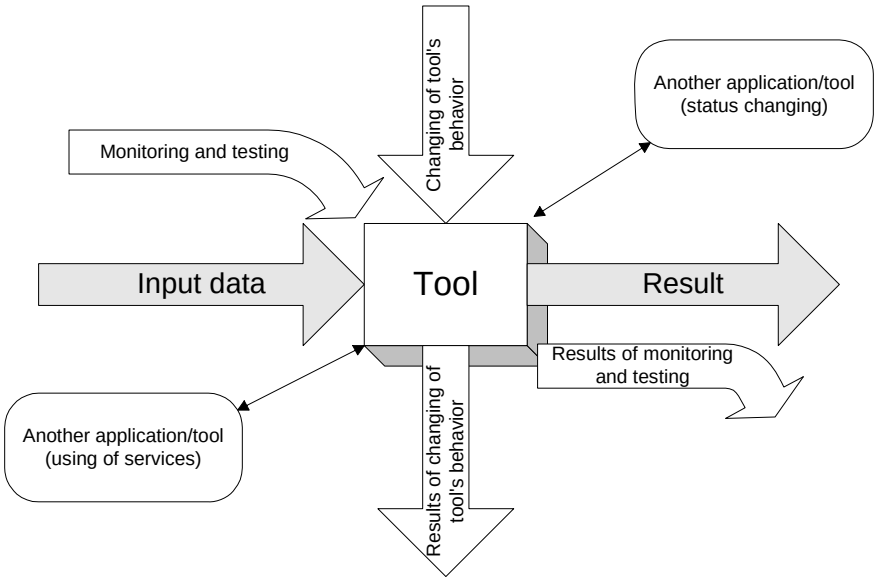

Figure 2 A conceptual architecture of a tool

7 Conclusions

The creating of tools' collections – environments is actual for many years [How82], and the importance of this topic only grows day by day [HOT00]. In this paper we briefly review some more important concepts and principles to organize (integrate) tools into one collaborating environment.

The main attention is paid to a subset of all possible environments – an information processing environment (information system) and an information system software environment that supports building of information system. There is a challenge for developers to create information system as a suit if tools.

The most important issues in this field are separation of concerns, integration and coordination, "plug and play", and support for multiple views. Traditional software development lifecycle is not acceptable for new emerging technologies, and researchers look for new methodologies. We consider that most of the tools in one environment have to built based on common principles, and that tool has to base on domain specific language (DSL) that describes tool behavior. In that approach the environment is a set of *interpreters* that interprets the tool specification (program in DSL) and each *interpreter* acts like desired tool.

8 References

[HOT00] W. Harrison, H. Ossher, and P. Tarr. Software Engineering Tools and Environments: A Roadmap. *Proceedings of the conference on The future of Software engineering (ICSE '00)*, pp.261-277, 2000.

[How82] W. E. Howden. Contemporary Software Development Environments. *Communications of the ACM*, 25(5):318-329, May 1982.

[Rei96] S. P. Reiss. Software Tools and Environments. *ACM Computing Surveys*, Vol. 28, N0. 1, March 1996.