

The Electronic Tool Integration Platform

Volker Braun, Tiziana Margaria, Bernhard Steffen

Universität Dortmund, Fachbereich Informatik

D-44221 Dortmund, Germany

Tel. +49 231 755.5801, Fax +49 231 755.5802

{volker,tiziana,steffen}@ls5.cs.uni-dortmund.de

Abstract

The paper presents the Electronic Tool Integration platform (ETI), a community platform designed for the project-specific, domain-specific, public or private interactive experimentation with heterogeneous tools. All groups of ETI users are assisted by an advanced, personalized Online Service guiding their activities, with particular attention to the experimentation, coordination and simple browsing of the available tool repository according to their degree of experience. This enables even newcomers to master the wealth of existing tools in a short timespan, and to identify the most appropriate collection of tools to solve their own application-specific tasks. Concepts and current realization are illustrated along a concrete instantiation: the ETI community associated with the International Journal on Software Tools for Technology Transfer (STTT).

1 The Goals

The Electronic Tool Integration platform (ETI) is designed for the support of communities wishing to set up sites for the project-specific, domain-specific, public or private interactive experimentation with heterogeneous tools¹. ETI managers, developers, tool providers, integrators, and users are assisted by an advanced, personalized Online Service guiding the platform's setup and the direct experimentation, coordination, and simple browsing of the available tool repository according to their role and degree of experience. In particular, this allows even newcomers to orient themselves in the wealth of existing tools and to identify the most appropriate collection of tools to solve their own application-specific tasks. Thus the ETI project can be seen as a concrete initiative that stimulates the communication between tool providers and tool users as well as between academia and industrial practice, supporting the transfer of tool-related technology.

¹The ETI platform is realized on top of the `METAFrame` environment [26, 20, 24].

ETI, born in 1996, is a Web-based, open communication platform, where tool providers can publish their tools and get valuable feedback from the end users, and users can compare different tools within their application domain. In particular, they can combine functionalities of tools of different application domains to solve problems a single tool never would be able to. A more detailed exposition, including background and related work can be found in [25], which describes the original instantiation for the ETI community associated to the International Journal on Software Tools for Technology Transfer (STTT) [11].

In this paper we present the ETI-approach while focussing on the ETI Online Community Service, an environment designed for users wanting to create their own application-specific or community-specific ETI sites. We also address the motivations and benefits of these sites, which are intended to constitute a meeting point for distributed communities for tool-related discussion and experimentation. In particular they are intended to spread new technologies at the tool-level, and to convince and teach potential users via easy experimentation. Central is here the ETI-sites' unique support for high-level tool coordination and for personalization and online-discussion. Details about the ETI Online Community Service can be found under <http://www.eti-service.org>, and the ETI-site for the Journal on Software Tools for Technology Transfer is accessible via <http://eti.cs.uni-dortmund.de>.

Each ETI-site plays, in fact, a public service role, giving users the possibility of direct, *hands-on, experience* with a wealth of available tools and functionalities. This also includes features like the *ETI Online Forum*, where users may e.g., propose case studies, and report on their experiences [25]. The associated online service is intended to develop into a *collaborative, independent tool presentation and evaluation site*: in the context of the service, users are invited to report on their experience with the integrated tools, as

- *directory* for possible tools and algorithms satisfying totally or partially their needs,
- (vendor- and producer-) *independent test site* for trying and comparing alternative products and solutions without any installation overhead,
- *quality assessment site* for the published tools, which are refereed according to requirements like originality, usability, installability, stability, performance, design,
- *independent benchmarking site* for performance on a growing basis of problems and case studies.

We are optimistic that the typical hesitation to try out new technologies can be overcome since serious hurdles, like installation of the tools, getting acquainted with new user interfaces, lack of direct comparability of the results and of performances, are eliminated. Moreover, the intended collaborative effort of the ETI user community to provide easily accessible information about fair, application-specific evaluations of various competing tools on the basis of predefined benchmarks, will be of inestimable help for

everybody in need of tool support. This is particularly important in geographical areas where support and distribution are not as favourably present.

In the following, after a summary of the related approaches (Sect. 2) and a brief history of the initiative (Sect. 3), we describe the ETI community Online Service (Sect. 4), present the ETI-Sites' meta-model (Sect. 5) and illustrate on a simple example its coordination features (Sect. 6). We then present the platform's multilayered software architecture (Sect. 7) and finally draw our conclusions and perspectives (Sect. 8).

2 Related Approaches

We are not aware of any project which combines the features provided by the ETI platform. Roughly, the approaches and tools coming closest to our platform can be classified as follows:

2.1 Approaches Focusing on Tool Access

- **Link collections:** There are a lot of Web sites available which provide links to software tools in a special application domain. Examples are the Petri Nets Tool Database [22] or the Formal Methods Europe [9] database.

The ETI platform covers this aspect of tool access by providing links to the home page of any tool available in the tool repository.

- **Software archives:** In contrast to software archives, like collections of Java Beans, ETI is not a distribution platform: tool providers retain all legal rights to, and responsibilities for their software. The user cannot download the software. He can only experience with the tools installed on one of the ETI application servers. To obtain a license of the chosen tool, the user can contact the tool provider via the information supplied by the platform.
- **Web sites providing execution facilities of a single tool:** Some Web sites like the HyTech home page [12] or the smv guided tour [23] give access to tool functionality via an HTML form. In most cases simple CGI [4] scripts are used to access a single functionality of the tool and to deliver the results to the user in text form.

This kind of tool access is also available within the ETI platform. In addition to that, users can combine tool functionalities to sequential programs and then execute them.

2.2 Approaches Focusing on Tool-specific Communication

- **News Groups and Mailing Lists:**

Mailing list like the Uppaal [17] user's list, or news groups provide a means for tool-specific communication, however, without any tool related functionality. They only address one aspect of the ETI platform, i.e. the communication functionality. Precondition to participate in the discussion are email and a news client.

- **Community Sites:**

Community sites like the Java Developer Connection [14] offer discussion groups, email notification, and information in a special application field. Like in the previous item, these community sites only cover the community functionality, however, in a more integrated fashion. They do not provide Web-based experimentation support.

2.3 Approaches Focusing on Tool Integration and Coordination

- **PROSPER:**

The PROSPER (Proof and Specification Assisted Design Environments) [7] toolkit provides an infrastructure where existing verification tools can be integrated into nearly any application (like CAD and CASE tools). This provides end users with applications enriched with new validation features, but it does not provide any user-level handle to coordinate verification tools. Coordination is exclusively done by the PROSPER expert.

- **ToolBus:**

Closest to ETI is TOOLBUS [2, 16], a very general environment for tool coordination. However, being based on a process algebra, its coordination specifications are much more general and complicated than in ETI. Thus the user community is rather restricted.

In contrast, the ETI project provides an open platform, where the tool coordination facility is designed to be available to a very wide community. To coordinate different tool features, experienced users may use ETI's coordination language HLL (High-Level Language). Unexperienced users are additionally supported by ETI's synthesis component. Here, the user can define *what* he wants to achieve, instead of specifying *how* he wants to achieve his goal. ETI's automated synthesis component then generates sequential programs that implement the intended task. Generated programs can be executed via the Internet.

3 A Brief History of the ETI Project and its Roots

- **1993**

Start of the METAFrame project [26, 20] at the University of Passau, Germany. In the initial phase an interpreter for the Pascal-like coordination language HLL (High-Level Language) and a graph library was implemented by four students.

- **1994**

The initial version of today's ETI's synthesis component was added to the METAFrame environment. It offers sophisticated support for the systematic and structured computer aided generation of application-specific complex objects from collections of reusable components.

- **03.1995 - 10.1995**

The first version of the Service Definition Environment was implemented in a cooperation between the University of Passau and Siemens Nixdorf Informationssysteme AG in Munich.

- **1996**

The ETI project was founded, in the context of the Springer International Journal on Software Tools for Technology Transfer (STTT) [11], with the goal to provide platform independent, Web-based access to METAFrame's coordination and synthesis functionality. It was intended to establish a virtual meeting point to organize the ETI community for flexible tool evaluation and coordination.

- **31.03. - 2.4.1998**

First official presentation of STTT/ETI at ETAPS' 98, the first European Joint Conferences on Theory and Practice of Software, in Lisbon, Portugal, including online ETI demonstrations.

- **1999**

Start of the platform re-engineering process, motivated by feedback of the platform users.

- **06.2000**

Launch of the ETI community site (<http://www.eti-service.org>) and of the current version of the ETI platform.

4 The ETI Community Online Service

The ETI Online Service is an environment designed for users wanting to create their own application-specific or community-specific ETI sites: These sites are intended to constitute a meeting point for distributed communities for tool-related discussion and experimentation. In particular they are intended to

spread new technologies at the tool-level and to convince and teach potential users via an easy experimentation.

Its Web site at <http://www.eti-service.org> organizes the collaborative effort to enhance the ETI platform by coordinating the quite heterogeneous and worldwide distributed ETI community. The ETI online community functions as an informal meeting point for the exchange of opinions: end users, tool providers, tool integrators, platform developers and ETI site managers are invited to comment, to query, and to provide feedback on the platform. The focus of this site is the generic platform which can be instantiated using tools from a chosen application domain. This application or community-specific instantiation is typically done at an ETI Site. After setup, ETI users can access the software tools provided by a site using the *ToolZone* client software, available on the site.

The functionalities offered by the ETI platform cover two distinct needs:

- features used to organize the ETI community like (tool-specific) discussion groups, newsletters, a developer's corner etc., and
- features that give direct access to the tools contained in the ETI site's tool repositories.

Together, these features allow a user to retrieve information on the available tools, execute single tool functionalities, combine tool functionalities, execute tool combinations, and exchange and discuss information and experiences with tool providers and other ETI users in a cooperative virtual space.

In the following, we will give some information about the community aspect. The subsequent sections will then discuss the central requirements and ETI-concepts while focussing on the coordination aspects.

4.1 The ETI Community Infrastructure

The ETI platform offers a virtual meeting point, where people involved in the development, use, and maintenance of the platform (called ETI community) may exchange information, and discuss on topics of their interest. This virtual meeting point is implemented as a Web application that provides access to mailing lists, discussion groups, frequently asked questions, software updates, documentation, etc. To ensure that everyone gets access to the right information in a fast and easy way, the application is customizable by the end users. Additionally, a profile-based permission concept controls the access to the offered functionality and data. Rather than presenting the community application itself, which can be easily accessed on the Web at the mentioned URLs, we prefer here to document the development process and the tool which have been used to build this component of the platform.

The ETI community application has been implemented and is continuously being extended, as mentioned in the brief project history, using the Service Definition Environment (SDE) [28, 27] of the METAFrame project. The SDE is a workflow design tool and it offers a compiler which generates a

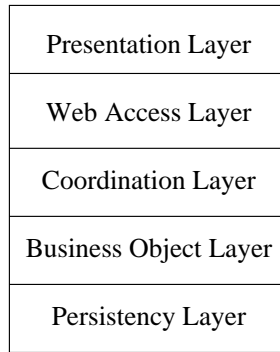


Figure 1: The Layers of a Web Application

Java Servlet [10] out of the workflow specifications (see [3] for more information). The main difference between this technology and other widespread techniques for Internet site development (like e.g. Allaire’s Cold Fusion Scripts [5] and Server Side Includes [6]) is the strict architectural separation of different layers with different purposes: as shown in Figure 1, we distinguish the application’s GUI (*presentation layer*), the application logic (*coordination layer*), and the base functionality implemented by the *business object layer*.

This clean structure has clear advantages wrt. ease of instantiation and of maintenance. To this aim, we introduce an explicit representation of the application logic as a directed graph, called Service Logic Graph, like the one shown in Figure 2, instead of embedding database queries or scripting fragments into the HTML code. Within the Service Logic Graph, nodes represent functional entities called Service Independent Building Blocks (SIBs) and edges represent the flow of control between the SIBs. This graph can be easily edited and modified without programming, simply using the service logic graph editor of the Service Definition Environment, as shown for a small service graph example in Figure 2.

As consequence, the service logic graph provides the developer with a *global* instead of a *local* (page-oriented) view of the application. This makes the development and the maintenance of the application easier. Using the service logic graph editor, the application can be configured without any programming effort and a new GUI may be designed without touching the application’s code.

In contrast to other known tools, the application can be analyzed already during its design using the validation features offered by the Service Definition Environment (see [24] for more details). These features are used to validate the application on the service logic level, which summarizes the application in a workflow-oriented fashion. Service logic graphs do not consider the implementation code of the SIBs, which are considered atomic entities and assumed to be correct. This has two reasons. Since a typical end user of the Service Definition Environment has little or no programming skills, he would be unable to interpret error reports on the implementation level. On the other hand, verification of complex Web applications on the code level is unfeasible if not undecidable.

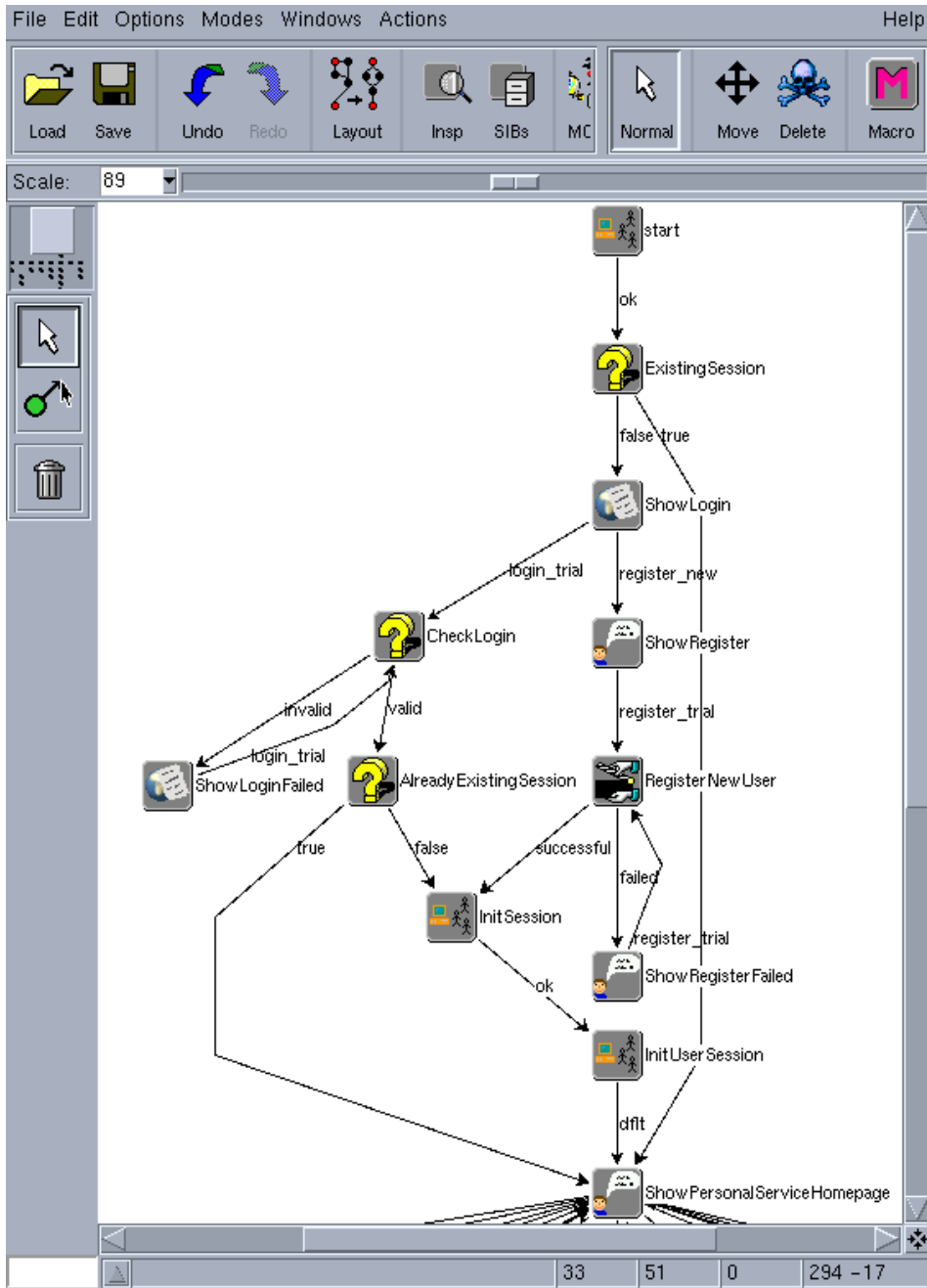


Figure 2: The Service Logic Graph Editor

In addition to the tool, we introduced a process which supports the development of reliable Web applications. One of the main aspects of this process is the collaborative organization of different user groups having different skills: HTML specialists build the GUI of the application, application experts configure the application using the Service Definition Environment, SIB integrators provide the SIBs and specialists having skills in object-oriented techniques design and implement the base functionality required to realize the SIBs. In combination with the validation features offered by the Service Definition Environment, this division of labor is our key concept for developing complex, reliable Web applications with a cooperative distributed effort.

4.2 Organizational Meeting Point

The ETI community site <http://www.eti-service.org> organizes the global effort to enhance and application-specifically instantiate the ETI platform. It serves as a virtual meeting point for the exchange of experiences and opinions: end users, tool providers, tool integrators, platform developers and site managers are invited to comment, to query, and to provide feedback about the platform concerning its potential to support ETI-instantiations. The actual application-specific instantiation is then done at one of the ETI-sites. These instantiations may then well be used for tool-related teaching and as a platform for potential customers looking for tools solving their current problem.

The way we model and conceptualize the treatment of complex tools is explained in the following section.

5 The ETI-Sites' Meta Model

The ETI meta model introduces the terms used in modelling and designing the repository of each ETI site's application domain, and defines the relations between them. Figure 3 shows a UML class diagram of this model, whose key notions are detailed in the following sections.

5.1 Activities

In general, tools are not integrated into the tool repository as monolithic blocks. Rather, single tool features are identified and prepared to be accessed via the platform. To this aim each tool feature is wrapped into an ETI-specific component called ETI-Activity, which constitute the elementary components used later for the tool coordination. An ETI-Activity models a tool feature as a *transformational* entity. This means that a tool feature can be seen as a component taking an object of type T_1 as input and delivering an object of type T_2 as output. Tool features beyond this kind of modelling can also be integrated but are formally treated as activities during our analyses.

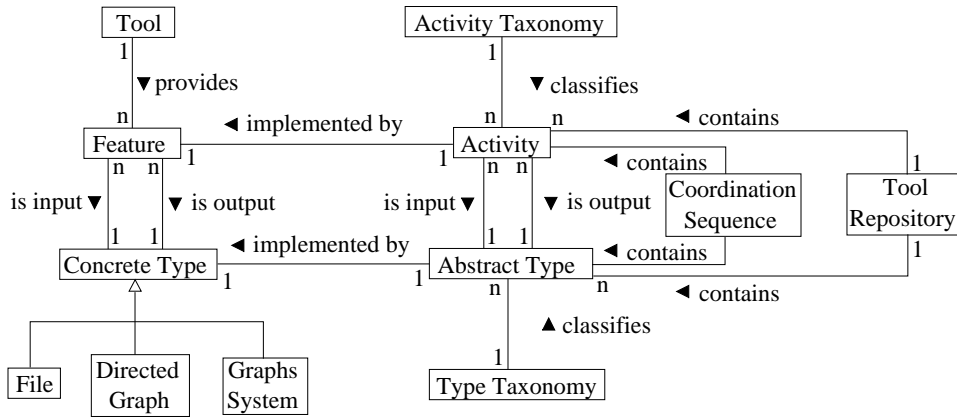


Figure 3: The ETI Meta Model

Each activity provides an interface, which defines the view of the platform on this component. In contrast to other component models, like DCOM [31] or JavaBeans [13], the structure of the interface of an activity is very simple: it contains only one operation reflecting the transformational character of the activity. Thus the signature of these operation consists only of an input type and an output type.

As an example, Table 1 shows a set of simple activities identified in the world of the UNIX operating system, including the associated text processing commands their implementation is based on.

Activity Name	Input Type	Output Type	Unix Cmd.	Description
latex	TEXFile	DVIFile	latex	Structured text formatting and type setting program.
dvips	DVIFile	PSFile	dvips	Converts a TeX DVI file to PostScript.
gv	PSFile	Display	gv	A PostScript viewer

Table 1: Activities Table for text editing-related commands

Up to this point, activities are just specified by one operation defining their input/output behavior, and types are just abstract names. This declarative view on an activity is sufficient for ETI's synthesis facility which combines single activities to sequential programs (see Section 6.1). However, in order to execute an activity, an implementation of the entities contained in the tool repository (i.e. the activities and the types) is required.

The implementation of a type is very simple: the (abstract) types found in the activity signatures are mapped to (concrete) data types used by the chosen tool feature. In the current status of the project, three generic data types are available: files, directed graphs and graph systems. These generic types cover the most important data types required in ETI's first application domain: the analysis and verification

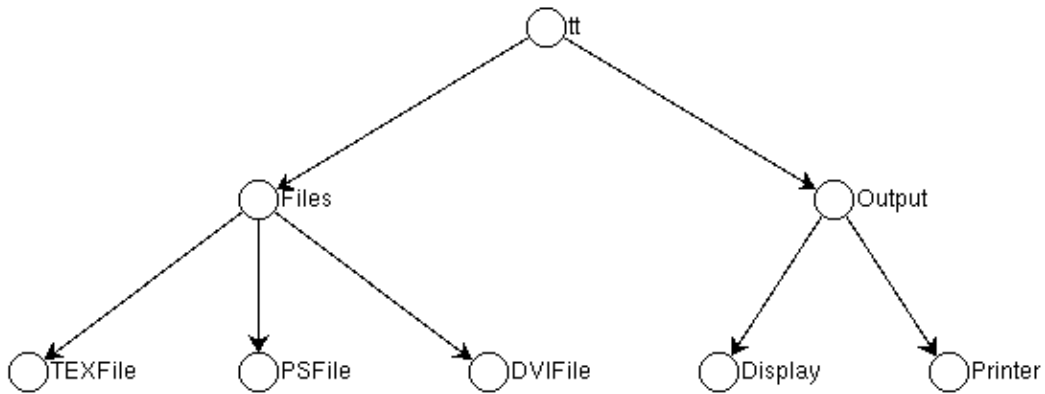


Figure 4: A UNIX-based Type Taxonomy

of distributed (real-time) systems.

Activities can be executed in two modes. In the stand-alone execution mode a single activity is run by the user. Activities can also be executed in the context of a coordination program. In this case, the activity is run in tool-coordination execution mode. Consequently, the implementation of an activity is done by providing two code fragments, each of them specifying the behavior of the activity in one execution mode.

5.2 Taxonomies

For a flexible handling (retrieval, loose object specification, abstract views), activities and types are classified by means of *Activity and Type Taxonomies*. Formally, a taxonomy (defined over a set of objects S) is a projection of the power-set lattice over S onto elements with a particular profile, which are identified by names.

Taxonomies can be represented as directed acyclic graphs (DAGs) (see Fig. 4), where each leaf represents an atomic entity (here activity or type) and each intermediate node represents a set of entities. Conceptually, edges reflect an *is-a* relation between their target and source nodes. From the semantics point of view, an intermediate node represents all atomic entities which can be reached from this node within the taxonomy DAG. With respect to this model, edges reflect the standard set inclusion.

Fig. 4 shows a simple classification of the types introduced in Table 1. Here, the type group `tt` represents all types which are available in the tool repository, i.e. `TEXFile`, `PSFile`, `DVIFile`, `Display`, and `Printer`. In contrast, the type group `Files` only represents the types `TEXFile`, `PSFile`, and `DVIFile`.

6 Coordination Sequences

On the basis of the available activities, users can build and then execute sequential programs called *coordination sequences*, which are finite paths of the form

$$T_1 \xrightarrow{a_1} T_2 \xrightarrow{a_2} T_3 \cdots T_{n-1} \xrightarrow{a_{n-1}} t_n$$

where T_i is a type and a_i is an activity which transforms an object of type T_i into an object of T_{i+1} . For illustration, we provide the concrete example of a UNIX-based coordination sequence, which uses the activities introduced in Table 1.

Example: A UNIX-based Coordination Sequence

The following coordination sequence takes a tex-file as input, runs the `latex` command on it, transforms the DVI result into a PostScript file and displays the PostScript file on the screen using the PostScript viewer `gv`.

$$TEXFile \xrightarrow{latex} DVIFile \xrightarrow{dvips} PSFile \xrightarrow{gv} Display$$

6.1 Loose Specifications

Building a coordination sequence manually is a non-trivial task: the user must know the activities contained in the tool repository, and must solve tool interfacing issues like finding the right data type transformer to connect features possibly provided by different tools. To ease this task, the ETI platform offers a synthesis component which automatically generates coordination sequences from abstract descriptions. These abstract descriptions are loose in two orthogonal dimensions:

- *Local Looseness*: Characterization of types and activities at the abstract level of taxonomies, instead of enumerating them explicitly. Here names contained in the taxonomies are interpreted as propositional predicates. They can be combined by means of the Boolean operators $\&$ (*and*), $|$ (*or*), and \sim (*not*) to specify sets of activities and types. With respect to the type taxonomy presented in Fig. 4, the formula $tt \ \& \ \sim \ Files$ characterizes the types `Display` and `Printer`.
- *Global/Temporal Looseness*: The characterization of whole coordination sequences is done in terms of abstract constraints specifying precedences, eventuality, and conditional occurrence of single taxonomy entities, rather than specifying the precise sequence of the types and activities. This kind of looseness is based on Semantic Linear-time Temporal Logic (SLTL) [29].

With respect to the activities shown in Table 1 and the type taxonomy presented in Fig. 4, the user could write a loose specification of the form of

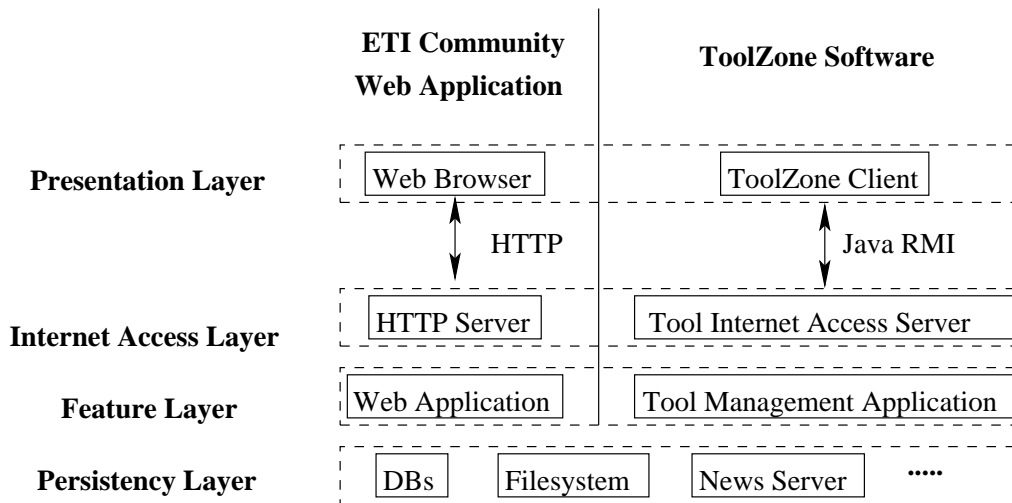


Figure 5: The logical Platform Architecture

TEXFile > Output

to query for all coordination sequences able to display a `TEXFile` on some `Output` device. In this example, ETI's synthesis component delivers the coordination sequence shown in the above Example as result. Note that the specification does not contain the exact description of the output device: we use the type group `Output` instead of one of the atomic types `Display` or `Printer` (local looseness). Additionally, the formula only specifies that an `Output` object must follow a `TEXFile` object without fixing when (as direct successor or anywhere later in the future) this occurrence should take place (temporal looseness). This is specified using the *before* operator `>`.

Using loose specifications, the user characterizes what he wants to achieve instead of how to achieve it. This goal-oriented aspect is the main difference between ETI's coordination facility and other coordination approaches like UNIX piped commands, scripting languages (e.g. Perl [32], Python [18]) or the ToolBus [2, 16]: there the user is forced to specify the exact coordination process.

7 The Platform's Software Architecture

Logically, each ETI-site's platform is built using a four layered software architecture (see Fig. 5).

The *persistency* (database) layer stores the tool repository as well as the discussion groups, mailing list archives, and user data. On top of this layer, the *feature* layer provides the real functionality offered by the ETI platform. This layer contains a Java-based Web application which implements the community services as well as a C++-based tool management application that gives access to the tool-related functionality. The *Internet access* layer makes the functionality offered by the platform accessible via

the Internet. For its implementation we use a standard HTTP server (like Apache [1]) to make the community services available to the public. The functionality provided by the tool management software is encapsulated by the tool Web access server which can be connected via Java RMI [8]. The *presentation* layer contains software components that enable the site users to access the functionality provided by the feature layer via an intuitive graphical user interface. Here, a standard Web browser can be used to participate in the community infrastructure. In contrast, a Java client (called ToolZone client) is required to gain access to the tools contained in the tool repository. This ToolZone client can be launched from the Web browser using the Java Web Start technology [15].

Usability, security, openness, and maintainability are the key requirements for a successful Internet service. In the ETI platform they are addressed as follows:

7.1 Usability

- Intuitive graphical user interface (GUI) for the client application: ETI provides an elaborate GUI with help functionality and hypertext support.
- Easy access to the features provided by the platform: The taxonomy-based retrieval mechanism together with the synthesis of coordination sequences from loose specifications enables a goal-oriented use of the platform without requiring any knowledge about the repositories' content and the interface-constraints between the integrated functionalities.
- Minimal set of requirements for client machines: The community features of the ETI platform can be used with a standard Web browser, and the access to the tool repository is provided by a Java-based software which requires a standard Java Runtime Environment to be installed on the client machine.
- Availability and performance: ETI distributes its functionality over a server farm. A load-balancing component ensures a performant access to the provided functionality.

7.2 Security

The ETI platform offers protected home areas and a profile-based security concept to control the access to personal data and functionalities. In addition, sensitive parts of the community features are only accessible using Secure Socket Layer (SSL) based encryption.

7.3 Openness

"A-posteriori"-integration is a key requirement for the ETI platform: every tool should be integrable into the tool repository as an ETI-activity. In fact, ETI makes no assumptions on the design, the availability

(e.g. communication via source code or binary version, operating system), and the accessibility (e.g. system calls, Java RMI [8], CORBA [21]) of the tool functionalities.

7.4 Maintainability

End users want easy, fast and reliable access to the functionality provided by the ETI platform. In particular, they want to

- browse through the tool repository looking for available activities,
- combine activities to coordination sequences and execute them using the ToolZone client,
- access tool-related information and discussion groups via a standard Web browser.

The corresponding maintenance of the ETI platform, which we support by means of the ETI community site, requires cooperation by people of different skills and profiles. We distinguish four groups of experts involved in different ways with the platform: tool providers, tool integrators, platform developers, and site managers.

Tool Providers

Tool providers are end users with special focus. Whereas the standard end user looks for candidate tools or a combination of tool functionalities to solve a certain problem, the tool provider is mainly interested in publishing his tool and getting valuable feedback by the end users. In addition to the tool itself, the tool provider supplies benchmarks, examples, and descriptions defining the tool's profile.

Tool Integrators

Tool integrators make new activities and types available to the ETI tool repository. They investigate the tools to be integrated, identify new activities, and establish connections between the new activities/types and already available ones. Thus they need a good understanding of the tools to be integrated and of the application domain modeled in the tool repository. In addition, they must be expert in C++, the language of the tool management application. We support the integration task by providing a tailored process together with some utility tools (see [3]).

Platform Developers

We distinguish two groups of platform developers. One group providing new SIBs, which can then be used to extend the ETI community application. An another group adding new functionality to the ToolZone software. Whereas SIB developers are only required to have Java skills, ToolZone software developers must have Java and C++ knowledge. In addition, ToolZone software developers need a good understanding of the platform's architecture.

Site Managers

Site managers build up and maintain an ETI site. This includes setting up the servers, installing the required software and customizing the site-specific version of the community application. They need expertise about UNIX-based operating system (e.g. Linux or SUN's Solaris) for the installation process. The customization of the Web application, which requires some fundamental HTML knowledge in order to adapt the look and feel of the Web site, is done with our Service Definition Environment.

8 Conclusions and Perspectives

We have presented the ETI community service, which allows users to create their own application-specific or community-specific ETI sites: these sites are meant to constitute a meeting point for distributed communities for tool-related discussion and experimentation. In particular they are intended to spread new technologies at the tool-level and to convince and teach potential users via an easy experimentation. Thus ETI-sites may well be used for tool-related teaching and as a platform for potential customers looking for tools solving their current problem.

As an example, the ETI Online Service for STTT currently comprises verification tools for real time systems and model checkers. The integration of programming language tools like type checkers, optimizers and code generators is on the way. The STTT ETI Service can be accessed via its homepage, <http://eti.cs.uni-dortmund.de>. From there, users can

1. **access online information** on the tools via hyperlinks to each tool's home site.
2. **access online a stand-alone version** of each tool, centrally located at the ETI service sites
3. **access the ETI repository** of integrated tools. It contains a collection of functionalities offered by individual tool, classified for ease of retrieval according to behavioural and interfacing criteria.
4. **experiment** at ease with the integrated tools and functionalities, by
 - (a) *running the* (stand-alone or integrated) *tools* on libraries of examples, case studies, and benchmarks made available on the ETI platform,
 - (b) *testing and running single tool functionalities*, capturing specific features offered by the integrated tools, on the same examples, from within a uniform graphical user interface provided by ETI,
 - (c) *constructing own application-specific heterogenous tools* through combination of single functionalities coming from different tools within the ETI platform,

- (d) *loosely specifying coordination tasks*, which can be then automatically completed by means of ETI's coordination support. This, in particular, takes care of data format incompatibilities, as detailed in [19].

5. **experiment with own sets of data**, to be deployed in user-specific, protected home areas.

We are optimistic that this will help overcoming the typical hesitation to try out new technologies: serious hurdles, like installation of the tools, getting acquainted with new user interfaces, lack of direct comparability of the results and of performances, are eliminated. Moreover, the intended collaborative effort of the ETI user community to provide easily accessible information about fair, application-specific evaluations of various competing tools on the basis of predefined benchmarks, will be of inestimable help for everybody in need of tool support. This form of remote assistance and advice is particularly important in geographical areas where support and distribution are still rare.

References

- [1] *The Apache Project*, <http://www.apache.org>.
- [2] J. A. Bergstra, P. Klint. *The ToolBus Coordination Architecture*, Proc. int. Conf. on Coordination Models and Languages, LNCS 1061, pp. 75-88, Springer Verlag, 1996.
- [3] V. Braun, T. Margaria, C. Weise: *Integrating Tools in the ETI Platform*, [30], pp. 31-48.
- [4] *CGI: Common Gateway Interface*, <http://www.w3.org/CGI/>.
- [5] *Cold Fusion, a cross-platform Web Application Server*, <http://www.allaire.com/products/coldfusion/>.
- [6] T. David, *Server Side Includes (SSI) Tutorial*, <http://www.tdscripts.com/ssi.html>.
- [7] L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, T. Melham: *The PROSPER Toolkit*, Proc. TACAS'2000, Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1785, pp. 78-92, Springer Verlag.
- [8] T. Downing: *Java RMI, Remote Method Invocation*, IDG Books Worldwide Inc., 1998.
- [9] *Formal Methods Europe*, <http://www.fmeurope.org>.
- [10] J. Hunter, W. Crawford: *Java Servlet Programming*, O'Reilly, 1998.
- [11] *International Journal on Software Tools for Technology Transfer*, <http://sttt.cs.uni-dortmund.de>.
- [12] *HyTech Home Page*, <http://www-cad.EECS.Berkeley.EDU/tah/HyTech/>.

- [13] *JavaBeans: The only Component Architecture for Java Technology*, <http://java.sun.com/products/javabeans/>.
- [14] *Java Developer Connection*, <http://developer.java.sun.com/developer/>.
- [15] *Java Web Start*, <http://java.sun.com/products/javawebstart/>.
- [16] P. Klint, P. Olivier. *The TOOLBUS Coordination Architecture: A Demonstration*, Proc. of 5th Int. Conference on Algebraic Methodology and Software Technology (AMAST'96), LNCS 1101, pp. 575-578, Springer Verlag, 1996.
- [17] K. G. Larsen, P. Petterson, W. Yi: *Uppaal in a nutshell*, Int. Journal on *Software Tools for Technology Transfer*, Vol. 1, pp. 134-152, Springer Verlag, 1997.
- [18] M. Lutz: *Programming Python*, O'Reilly, 2001.
- [19] T. Margaria, V. Braun, J. Kreiler: *Interacting with ETI: A User Session*, [30], pp. 49-63.
- [20] T. Margaria, B. Steffen: *Coarse-grain Component Based Software Development: The METAFrame Approach*, Proc. STJA'97, "Smalltalk und Java in Industrie und Ausbildung", 10.-11. September 1997, Erfurt (D), ISBN 3-00-001828-X, pp. 29-34.
- [21] OMG. *The Common Object Request Broker: Architecture and Specification Revision 2.0*, Technical Document ptc/96-08-04.
- [22] *Petri Nets Tool Database*, <http://www.daimi.aau.dk/PetriNets/tools/db.html>.
- [23] *SMV guided tour*, <http://www.cs.cmu.edu/modelcheck/tour.html>.
- [24] B. Steffen, T. Margaria: *METAFrame in Practice: Design of Intelligent Network Services*, in LNCS State-of-the-Art Survey *Correct System Design Recent Insights and Advances*, Lecture Notes in Computer Science 1710, pp. 390 - 415, 1999.
- [25] B. Steffen, T. Margaria, V. Braun: *The Electronic Tool Integration platform: concepts and design*, [30], pp. 9-30.
- [26] B. Steffen, T. Margaria, A. Claßen, V. Braun: *The METAFrame'95 Environment*, Proc. CAV'96, Int. Conf. on Computer-Aided Verification - Juli-Aug. 1996, New Brunswick, NJ, USA, LNCS 1102, pp. 450-453, Springer Verlag.
- [27] B. Steffen, T. Margaria, A. Claßen, V. Braun, N. Kalt: *Hierarchical Service Definition*, Annual Review of Communications, Int. Engineering Consortium (IEC), pp. 847-856, 1997.
- [28] B. Steffen, T. Margaria, A. Claßen, V. Braun, M. Reitenspieß, M. Wendler: *Service Creation: Formal Verification and Abstract Views*, 4th Int. Conf. on Intelligent Networks (ICIN'96), pp. 96-101, 1996.

- [29] B. Steffen, T. Margaria, B. Freitag: *Module Configuration by Minimal Model Construction*, Techn. Rep. MIP-9313, Fak. für Mathematik und Informatik, Universität Passau (Germany), 1993.
- [30] *Special section on the Electronic Tool Integration Platform*, Int. Journal on *Software Tools for Technology Transfer*, Vol. 1, Springer Verlag, November 1997
- [31] T. Thai: *Learning DCOM*, O'Reilly, 1999.
- [32] L. Wall, T. Christiansen, J. Orwant: *Programming Perl*, 3rd Edition, O'Reilly, 2000.