

# Tool Interfacing Mechanisms for Programming-for-the-Large and Programming-for-the-Small

Yun Yang

Centre for Internet Computing and E-Commerce

School of Information Technology

Swinburne University of Technology

PO Box 218, Hawthorn, Melbourne 3122

Australia

E-mail: [yyang@it.swin.edu.au](mailto:yyang@it.swin.edu.au)

## Abstract

*Software development needs to be supported at both the organisational process level (programming-for-the-large) and the detailed coding level (programming-for-the-small). It is critical to seal the gap in order to enable effective software integration at both levels in a uniform manner instead of in isolation now. This paper is aimed at addressing fundamental issues related to integration techniques for productive software development. In particular, we focus on tool interfacing mechanisms for universally accessible data integration and plug-and-play tool integration to support software development uniformly.*

## 1. Introduction

We define programming-for-the-large as software development at the organisational level and programming-for-the-small as software development at the coding level. In this paper, software development at the organisational level is mainly aiming at process support and software development at the coding level is mainly aiming at software component support. In this context, we focus on research into new software development technologies for supporting *both* programming-for-the-large and programming-for-the-small.

Over the past decades of software development research, we have witnessed significant achievements at both the organisational and coding levels. At the organisational level, which we regard as programming-for-the-large, software development paradigms have evolved from (hard-wired/fixed) office automation systems to workflow-oriented generic process-centred environments. At the coding level, which we regard as programming-for-the-small, software development paradigms have evolved from structured programming, object-oriented

programming to component-based programming for individual tasks. In practice, process-centred environments and task-specific tools can play complementary roles. On the one hand, process-centred environments can be developed and viewed as cooperation of integrated individual tools composed of components supported by universally accessible data. On the other hand, process-centred environments can be used to orchestrate the interactions among tools and to support development and integration of tools where a process can be applied. Unfortunately, research into software development technologies for programming-for-the-large and programming-for-the-small has been largely disconnected. Our research addressed in this paper is targeted at sealing the gap between them.

In the real world, processes are an essential element in any workplace organisation. Generally speaking, a process is normally composed of partially ordered steps to reach a goal [1]. In order to get the work done in a well-organised manner, programming-for-the-large can be facilitated to support computer-mediated processes. Process-centred environments are software to support the management/coordination of steps in processes. Research into processes has been carried out intensively by several communities such as software engineering (for software processes) and information systems (for business processes). Those processes have many commonalities yet some differences [3]. Since this paper focuses on general software development technologies, we do not distinguish these similar processes explicitly here.

When we go down to the step level in a process, in most cases, each step contains a task being carried out by some people (team members) with support from task-specific software, such as an editor tool. In order to improve the software productivity, programming-for-the-small can be facilitated to support rapid software development based on components. By component, we mean that it is a reusable piece of software denoting a self-contained entity (black-box) that exports functionality to

its environment and may also import functionality from its environment using well-defined and open interfaces [10].

With the above description, it is clear that in reality, we rely on computer software more and more to carry out our work at both the organisational process level and the individual step level which correspond to programming-for-the-large and programming-for-the-small respectively. In this highly competing world, it is critical for software developers to deploy new technologies for more productive software development. Therefore, it is an important research area to investigate effective software development technologies for supporting both processes and individual tasks.

This paper is organised as follows. In the next section, we describe the related work briefly. In Section 3, we focus on addressing the corresponding novel tool interfacing mechanisms to support data and tool integration for programming-for-the-large and programming-for-the-small. After that, in Section 4, we illustrate our prototype re-engineering to support our research outcomes. In Section 5, we outline our next step for further evaluation. Finally, we draw the conclusions.

## 2. Related work

In this paper, key mechanisms explored are related to the tool interface for integrating data and tools in order to seal the gap between programming-for-the-large and programming-for-the-small. To elaborate, we look at these two issues one by one as follows.

### 2.1. Data integration

From the viewpoint of data integration of the tool interface, today the most popular data repository used for programming-for-the-large is based on a variety of databases [8, 19]. At the same time, the data format for programming-for-the-small varies even wider using such as different kinds of file format and database. It is very difficult to effectively exchange and/or share data among interoperable process-centred environments and task-specific tools. To solve the problem, the ideal case is to have universally accessible data for both programming-for-the-large and programming-for-the-small. This is the first key research topic addressed in this paper as initially discussed next.

We believe that tomorrow's Web and Internet will use XML (extensible markup language), its family languages, and the like. XML, first released in 1998, is sometimes called portable data. It has been developed by the World Wide Web Consortium ([www.w3c.org](http://www.w3c.org)) to encode information with meaningful structure and semantics that computers can readily understand. With this kind of portable data, the nature of software development may be fundamentally transformed. In particular, we facilitate

XML as a uniform data format not only for processes, but also for all task-specific tools. This will enable software to be interoperable to a much greater extent from the data integration perspective. At this stage, some initial work has been carried out on applying XML to programming-for-the-large and programming-for-the-small in isolation. For example, WfXML from Workflow Management Coalition ([www.wfmc.org](http://www.wfmc.org)) is used for proposing some workflow interoperability standards [5]. Some coding level examples are work presented in cXML ([www.cXML.org](http://www.cXML.org)) and XBRL ([www.XBRL.org](http://www.XBRL.org)).

### 2.2. Tool integration

From the viewpoint of tool integration of the tool interface, we note that in the literature, there exist reports on Integrated Development Environments (IDEs) such as SoftBench from HP and many others surveyed in [4] as well as our early work [12]. They are in effect at the low level of tool integration. In terms of software development, products of programming-for-the-large and programming-for-the-small should all be treated in a similar manner as software tools. Unfortunately, tools are still often implemented directly with lower-level programming languages. In addition, it is also very difficult to realise the relationship among programming-for-the-large and programming-for-the-small with old-fashioned tool integration. To solve the problem, the ideal case is to have portable code for plug-and-play tool integration supporting both programming-for-the-large and programming-for-the-small. This is the second key research topic addressed in this paper as initially discussed next.

We believe that tomorrow's applications and tools for processes and individual tasks will be dominated by plug-and-play of portable (services) tools, which involves modelling enterprises as collections of services. To support plug-and-play, component-based software integration is desirable. In general, components are the smallest self-managing, independent, and useful parts of a system that works in multiple environments. With many lessons learnt from component-based development [11], components seem to promise rapid application development and a high degree of customisability for end-users, leading to fine-tuned applications that are relatively inexpensive to develop and easy to learn [7]. We need to facilitate component-based development, for both programming-for-the-large and programming-for-the-small.

## 3. Tool interfacing mechanisms

In this section, we discuss the corresponding novel software development mechanisms on how to effectively interchange and share data/information and seamlessly integrate software tools for both programming-for-the-large and programming-for-the-small which are researched

into primarily in isolation at the moment. We note that this paper focuses on investigating unified tool interfacing mechanisms with facilitating portable data, e.g. XML, and portable code, e.g. components. It is not the intention of this paper to research into the XML language or the component-based software development paradigm themselves.

Some tool interfacing principles from our earlier work [14, 16, 18] are relevant and have been improved for this purpose in order to integrate data and tools for programming-for-the-large and programming-for-the-small. We address the inter-related data and tool integration mechanisms within the tool interface context as follows in this section.

### 3.1. Data integration mechanisms

For data integration, we focus on investigating unified portable data for both programming-for-the-large and programming-for-the-small. In this case, we use the XML family as indicated earlier. XML's power derives from its extensibility and ubiquity.

Although XML is well structured for encoding semantically meaningful information, it must be based on ontology. As ontology varies from domain to domain, and is dynamic for dynamically formed domains, the most significant issue is to exchange the semantics of domain models, and interpret messages differently in different problem domains. Sometimes data integration can be done by extracting information and resolving the conflicts [2]. Although it has its great potential, XML has already had problems caused by the growing number of XML industry vocabularies – each industry defines a closed standard unique to that industry. Therefore, standards are essential for interoperability.

Generally speaking, domain ontology provides a set of concepts, or meta-data, that can be queried, advertised and used to control the behaviour of tools. These concepts can be marked using XML tags, and then a set of commonly agreed tags that underlie message interpretation.

We base our work on identifying the gap between programming-for-the-large and programming-for-the-small in order to develop XML ontology for supporting both of them from the data integration perspective. On the top, our unified XML representation has a generic programming-for-the-large definition to handle process related information such as messages for coordination. In the middle, it has a bridge as interface to incorporate a set of bottom level domain-specific programming-for-the-small definitions to handle such as document data and internal information.

At the moment, we also work on innovative ontology engineering, supported by RDF (Resource Description Framework) based languages and tools.

### 3.2. Tool integration mechanisms

For tool integration, we focus on how programming-for-the-small realises programming-for-the-large and how programming-for-the-large helps programming-for-the-small in a complementary manner. Component-based software development in a plug-and-play manner is a cost-effective way of constructing tools because of its potential of high-level reuse.

To understand the relationship between programming-for-the-large and programming-for-the-small with respect to tool integration, we can illustrate as follows. Firstly, software components as programming-for-the-small, their development is a process which can be supported by programming-for-the-large more effectively. Secondly, as a process-centred environment, it can be constructed by components. Finally, due to the nature of software components, it is clear that the software development process may likely shift from the “specify, design, and implement” concept towards “select, evaluate, and integrate”. Therefore, ideally, selected components can be evaluated, for example, via introspection, before integration to construct tools for both programming-for-the-small and programming-for-the-large. This kind of process itself can also be supported by process-centred environments for effective plug-and-play.

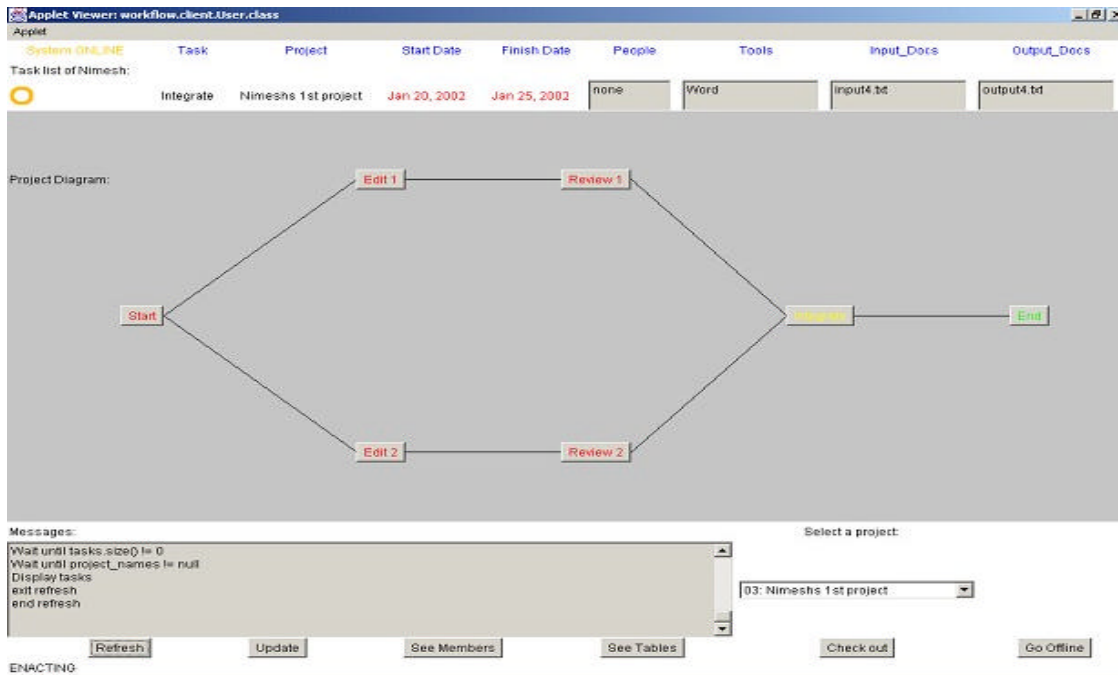
It is extremely important that component services are provided through a standard, published interface to ensure interoperability. Such components rely on robust distributed object models to maintain transparency of location and implementation. Since components can be distributed dynamically for reuse across multiple applications and heterogeneous computing platforms, the later characteristic is why Java (“write once, run anywhere”) portability has had such a dramatic impact on enterprise computing and component development, and why XML is essential for developing a shared Internet file system. A key issue here for tool integration is true software reuse. Again, it is clear that as software developers proceed to build the next generation systems, some components can be built and some components (COTS - commercial-off-the-shelf) can be purchased. The component repository grows gradually so that at a later stage systems can be rapidly constructed in a very cost-effective fashion.

## 4. Prototyping

For research on sealing the gap between programming-for-the-large and programming-for-the-small, some of our previously existing prototypes have been used for prototyping to support research described in this paper. The prototypes, all written in Java, include process coordination management and visualised process enactment [19,17] for programming-for-the-large and a

cooperative editor for programming-for-the-small [20]. These prototypes serve well as a base for prototyping to

support integrated programming-for-the-large and programming-for-the-small paradigms.



**Figure 1. Process enactment user interface**

Our process-centred environment prototype is based on the typical client/server architecture as described in [15]. Figure 1 shows a screen shot of the Java applet for a high-level simplified process-centred scenario enabled by the prototype which is explained next. For teamwork coordination in our environment, once the process is started, the most essential facility is that each team member is provided with a dynamic up-to-date *to-do* list for indicating the tasks to be done. For example, as depicted in Figure 1, the “integration” task, using the cooperative editor, is on the to-do list for that particular person. With notification, there could be other information to be passed on such as instructions/messages for the work and sensitive indicators for deadlines. Team members can use local tools, or tools available in the online teamwork environment, to carry out tasks such as taking the input documents and generating output documents. Sometimes, tools can and need to be specified in the process. When a certain task is completed, a notification is sent to notify the process engine so that the decision-making policy is utilised to generate new to-do lists for all affected team members. For a team member working in a team environment, it is very useful to have a global view of the process in a visualised fashion in order to create a better teamwork atmosphere as shown in Figure 1. This is important from the psychological point of view when a person works in a computer-mediated teamwork

environment. Different colours are used for status of each task to indicate whether a task is enacted, enacting or unenacted. By enacted task, we mean that the task is completed. By enacting task, we mean that the task is currently ongoing but has not been completed. By unenacted task, we mean that the task has not been invoked yet. The global view of the process is adjusted automatically whenever the status of any task is changed.

In order to demonstrate and evaluate our tool interfacing mechanisms proposed in the previous section, the existing process-centred environment prototype has been re-engineered so that it is based on:

- XML files, instead of a relational database used in the original environment, as the universally accessible portable data repository for data integration, and
- components in JavaBeans, instead of Java objects used in the original environment, as the reusable and portable code for tool integration.

For the evaluation purpose, the process-centred environment prototype with a relational database as data repository for storing process information has been re-engineered with purely using XML. All the tables in the database as detailed in [19] have been replaced by XML, which is relatively straight forward based on support of the XQuery engine ([www.fatdog.com](http://www.fatdog.com)). For example, the process XML file contains all the processes with the

process ID number and name; the task XML file contains the corresponding process ID number, its own task ID number, name and other task related attributes such as status; the dependants XML file contains the task ID number and its corresponding dependent process and tasks ID numbers. Certainly there are other XML files to contain information for each task of tools to be automatically invoked, team member to be involved, input and output documents to be used and generated, and so forth. As an example for the programming-for-the-small, we simply use our cooperative editor for a XML document using unstructured representation.

Furthermore, for the evaluation purpose, the process-centred environment prototype written in Java has been re-engineered with the following components developed. The front-end of the client side has a set of components on the top for the graphical user interface of processes such as for the to-do list, the attribute list, the graphical layout of the current process, and so forth. It also has a user logic component in the middle to handle process tasks. Then the components for message communication between the client and server sides are common, i.e. reusable by both sides. The server side also has a coordination component in the middle as process engine, which is sufficient for the evaluation purpose. In addition, at the bottom of the server side, there is a data manager component in order to access information in the back-end data repository. With the above components, the new re-engineered process-centred environment has been easily constructed. However, there are two issues we need to mention in particular. Firstly, the data manager itself is abstract which needs to be supported by a concrete data manager depending on the data source. For example, for a data source of a relational database, we have a JDBC data manager and for a data source of XML files, we need a XML data manager. This also makes our process-centred environment very flexible with dealing with all different data sources. Secondly, the process-centred environment cannot be constructed by simply putting components together. Some extra work needs to be done to glue the components. For example, we have to provide task data as non-component part for the user logic component. However, our experience shows that we can build up the process-centred environment based on components with minor extra work. We did not bother to re-engineer a particular programming-for-the-small tool using components since it would not contribute to the evaluation as the component-based process-centred environment itself can certainly be viewed as a tool.

Based on the above re-engineering work of the tool interface for both data integration and tool integration, the demonstration is delivered very successfully for the purpose of proof of concept of what we proposed in the previous section. In general, for the team members, they would not notice any differences, i.e. the re-engineered environment behaves the same as the original

environment. For the developers, data integration is now based on the portable data with XML and tool integration is now based on the portable code with components.

## 5. Further evaluation

Given the relationship between processes and individual tasks, we have used our process-centred environment, which is based on a client/server architecture, as backbone for supporting integration of programming-for-the-large and programming-for-the-small. However, the client/server architecture is not necessarily the only architecture and even not necessarily the best architecture for programming-for-the-large. Therefore, we tend to further evaluate our research outcomes based on a different environment architecture addressed in this section.

From the process point of view, we look at the framework for programming-for-the-large and how to support it. Process research literature can be found, for example, in many software engineering conferences and journals, particularly in 1990s. However, traditional workflow systems have been used mainly in large organisations because they can afford to install, run and maintain the systems. In addition, most process-centred environments have been criticised because they inflexibly prescribe temporal step sequencing, and narrowly dictate and restrict, rather than broadly assist, people in the roles they play. Barriers to deployment of process-centred environments in the past, such as lack of flexibility, failing to agree on standards, imposing incompatible repositories, and lack of using the Internet, have hindered the deployment of process-centred environments. Nowadays, the world is witnessing profound changes under the influence of Internet technologies [13]. At the moment, Web-based process-centred environment area is perhaps the most watched one for process support. However, most Web-based process-centred environments are based on the multi-tiered client-server architecture [6, 19] with a heavy-weight inflexible nature. So it is necessary to improve the current status.

It is believed that the next generation of process-centred environments needs to have at least the following characteristics: standardised information interchange formats, workflow/process services fully integrated with other services, and the Internet. In this section, we explore the wide area workflow to extend the traditional workflow capabilities with the above-mentioned characteristics. For example, users can participate in a process using integrated applications with interchangeable data. Via supporting tools, virtually any electronic medium, including fax, telephones, email, pagers, Web, PDAs (personal digital assistants) and so forth, can serve.

Since processes are an essential element in any workplace organisation, they need to be managed properly.

In this Internet era, process management is becoming an increasingly important part of organisational information systems. Attempts to develop process-centred environments have failed due to commonly-used heavy-weight inflexible frameworks. To overcome this long-puzzling problem of inappropriate frameworks, we believe that it is essential to have a light-weight dynamic architecture to reduce the cost of system running and increase the flexibility of process support. In our paradigm, we take process as the backbone to investigate both programming-for-the-large and programming-for-the-small in a complementary manner in order to establish an appropriate light-weight dynamic programming-for-the-large framework with comprehensive support from programming-for-the-small.

In this section, we focus on some fundamental issues related to the technologies of programming-for-the-large with wide area workflow which changes the way workflow is managed within single organisations as well as across the virtual enterprise with multiple organisations involved.

Defining an appropriate framework for programming-for-the-large is very important. To have a cost effective and practical process-centred environment, it is essential to design a light-weight dynamic framework for process support. We tend to facilitate, say, the peer-to-peer distributed architecture [9]. This needs a new programming-for-the-large paradigm for a process support architecture that (1) does not imply the presence of a heavy-weight centralised process engine for coordination, and (2) allows incomplete dynamic process to be specified. To realise this new architecture, process coordination needs to be based on negotiation when necessary between related software components of the environment with the capability of human interaction. With the support of negotiation, coordination can be carried out in a distributed fashion instead of controlled by a centralised process engine in the existing prevalent architectures. At the same time, it is important to allow human interaction for the purpose of some control and fine-tuning for coordination. To realise this new architecture, the framework should also support processes by enabling configuration of the process dynamically on the fly, i.e. the process can be constructed incrementally during enactment/execution when necessary rather than all details have to be pre-defined in the first place which is often impractical. Based on the above, the new framework can provide very flexible support to users by allowing them to do things they would like to do rather than to restrict users inflexibly by not allowing them to do things they wish to do. This system architecture would make process-centred environments light-weight and dynamic.

In general, research on the conceptual framework based on the paradigm of programming-for-the-large, with support from the paradigm of programming-for-the-small, will lead to the next generation of process-centred

environments which provides the flexibility and sophistication necessary to manage dynamic processes. To further evaluate the existing research outcomes, it would be good to conduct more prototyping to accommodate the new programming-for-the-large architecture and framework proposed in this section. This is our next step for this research work. However, we believe that, in general, it should not result in a significant change for the programming-for-the-large part and should have no impact on the programming-for-the-small part.

## 6. Conclusions and future work

The major outcomes described in this paper are the technologies for integrating software development at both the organisational process level (programming-for-the-large) and the detailed coding level (programming-for-the-small). In doing so, we have proposed integration mechanisms of both programming-for-the-large and programming-for-the-small in a uniform manner. More specifically, data integration is based on XML as portable data repository for both programming-for-the-large and programming-for-the-small and tool integration is based on software components as portable code which is the foundation for constructing process-centred environments and task specific tools whilst process-centred environments can serve for processes of components development and assembling.

In addition to the existing outcomes based on the traditional process-centred environment architecture, a light-weight dynamic wide area workflow framework for processes has been proposed in order to conduct further evaluation as the next step. We need also pay special attention on further development of XML, components, and the like, to improve the support of data and tool integration for programming-for-the-large and programming-for-the-small in a long run.

## Acknowledgement

This work is partly supported by Swinburne Vice Chancellor's Strategic Research Initiative Fund 2002-4 and Swinburne Research Development Grant 2002. I gratefully acknowledge Volker Gruhn for some discussions earlier. I also thank Glenn Ludlow and Nimesh Dedhia for the prototype re-engineering work involved.

## References

- [1] P. H. Feiler and W. S. Humphrey. Software process development and enactment: concepts and definitions. In *Proc. of 2nd Int. Conf. on Software Process*, pages 28-40, Berlin, Feb. 1993.

- [2] D. Fensel, Y. Ding, B. Omelayenko, E. Schulten, G. Botquin, M. Brown and A. Flett. Product data integration in B2B e-commerce. *IEEE Intelligent Systems*, 16(4):54-59, 2001.
- [3] V. Gruhn. Software process management and business process (re)engineering. *Software Process Technology, Lecture Notes in Computer Science*, 913:250-253, Springer, 1994.
- [4] J. C. Grundy and J. G. Hosking. *Software Tools*, Wiley Encyclopaedia of Software Engineering, 2nd Edition, Wiley InterScience, Dec. 2001.
- [5] J. G. Hayes, E. Peyrovian, S. Sarin, M-T. Swenson and R. Weber. Workflow interoperability standards for the Internet. *Special Issue on Internet-based Workflow, IEEE Internet Computing*, 4(3):37-45, 2000.
- [6] Y. Kim, SH Kang, D. Kim, J. Bae and K-J Ju. WW-FLOW: Web-based workflow management with runtime encapsulation. *Special Issue on Internet-based Workflow, IEEE Internet Computing*, 4(3):55-64, 2000.
- [7] S. M. Lewandowski. Frameworks for component-based client/server computing. *ACM Computing Surveys*, 30(1):3-27, 1998.
- [8] F. Maurer, B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kotting and M. Schaaf. Merging project planning and Web-based dynamic workflow technologies. *Special Issue on Internet-based Workflow, IEEE Internet Computing*, 4(3):65-74, 2000.
- [9] M. Parameswaran, A. Susarla and A. B. Whinston. P2P networking: an information sharing alternative. *IEEE Computer*, 34(7):31-38, 2001.
- [10] F. Plasil and M. Stal. An architectural view of distributed objects and components in CORBA, Java RMI and COM/DCOM. *Software - Concepts & Tools*. Springer-Verlag, 19(1):14-28, 1998.
- [11] M. Sparling. Lessons learned through six years of component-based development. *Communication of the ACM*, 43(10):47-53, 2000.
- [12] J. Welsh and Y. Yang. Integration of semantic tools into document editors. *Software - Concepts and Tools*. Springer-Verlag, 15(2):68-81, 1994.
- [13] A. B. Whinston. Electronic commerce: a shift of paradigm. *IEEE Internet Computing*, 1(6):17-19, 1997.
- [14] Y. Yang. Tool interfaces for software development. PhD thesis, The University of Queensland, Brisbane, Australia, 1992.
- [15] Y. Yang. Issues on supporting distributed software processes. *Software Process Technology, Lecture Notes in Computer Science*, 1487:243-247, 1998.
- [16] Y. Yang and J. Han, Classification of and experimentation on tool interfacing in software development environments. *Proc. of Asia-Pacific Software Engineering Conf. (APSEC'96)*. IEEE Computer Society Press, pp56-65, Seoul, Korea, Dec. 1996.
- [17] Y. Yang and P. Wojcieszak, Visual programming support for coordination of Web-based process modelling. In *Proc. of 11th Int. Conf. on Software Engineering and Knowledge Engineering*, pages 257-261, Kaiserslautern, Germany, June. 1999.
- [18] Y. Yang. Tool integration in a process-centred Web-based teamwork support environment in Java. In *Proc. of Australian Software Engineering Conf. (ASWEC'00)*, pages 215-220, Canberra, Australia, Apr. 2000.
- [19] Y. Yang. An architecture and the related mechanisms for Web-based global cooperative teamwork support. *Int. Journal of Computing and Informatics*, 24(1):13-20, 2000.
- [20] Y. Yang, C. Sun, and Y. Zhang and X. Jia. Real-time cooperative editing on the Internet. *IEEE Internet Computing*, 4(3):18-25, 2000.