In Proceedings of Australian Software Engineering Conference (ASWEC 2006).

In Proceedings of the 2006 Australian Conference on Software Engineering. Sydney, Australia 2006.

A Service-Oriented Architecture for Software Process Technology

Therese Helland
Computas AS
PO Box 482
1327 Lysaker, Norway
the@computas.com

John Grundy[1, 2] and John Hosking[1]
[1]Dept. of Computer Science and
[2]Dept. of Electrical and Computer Engineering, University of Auckland
Private Bag 92019, Auckland, New Zealand
{john-g, john}@cs.auckland.ac.nz

Abstract

As teams become more distributed and software systems increasingly complex, the difficulty of coordinating development processes becomes significant. Software Process Technology supports the planning and execution of software development processes to be managed and supported by computer programs. This paper describes the development of a decentralised process support and management tool, and explores its functionality for both process modelling and enactment. Our focus is on a highly distributed and service-oriented approach with the intention of providing good distribution of system components and easy integration of third-party tools and remote services for coordination by workflow.

1 Introduction

Software process technology supports the planning and execution of software processes to be managed and supported by a computer program [1, 3, 6, 22]. The process tool enables assignment of work to humans or other programs, passes the work on, and tracks its progress. An important potential advantage of such systems is the increased efficiency of software processes, resulting from the automation of previously manual and time-consuming tasks. Several different facilities are essential in providing computerised process support. Firstly, a means of defining processes and their execution is required. This is normally provided by a process modelling language and tool. Furthermore, functionality to allow for computerised enactment of the process model is typically provided by a process enactment engine. Other important features include good distribution of system components, seamless integration with third-party tools, and sufficient support for cooperative activities. Many current approaches to software process technology suffer from architectures with reliance on centralised servers, need for users to understand complex rule-based specification systems, limited feedback to users of process state, and limited accessibility of the coordination tools.

This paper describes the design and implementation of a prototype software process support tool, IMÅL, which aims to address some of the weaknesses we have identified with previous approaches. The IMÅL environment employs a service-oriented architecture in order to provide better distribution of tool components and allow easier, more dynamic integration of environment components. It uses a user-tailorable meta-tool to support process specification and state visualisation, allowing run-time modification of the process visualisation notations. Integration of third-party document management and complex rule processing systems via a service-oriented approach devolve these tasks to other applications.

We first describe the motivation for this research and critique some other existing process technology tools. We give an overview of IMÅL and our process modelling notation used in the tool. We illustrate a simple process example, describe the tool's architecture and illustrate its usage. Finally, we evaluate IMÅL's effectiveness for decentralised process support and give some directions for future research.

2 Motivation

Consider a software team collaborating on a software modification task. The team may want to make changes to the design of a set of modules, review these proposed changes, allocate implementation of the changes amongst the team, and carry out various unit and integration tests to validate the changes. The team may be working in a co-located or distributed fashion. Many researchers and practitioners have identified the problems in coordinating such work, including managing access to shared documents (design, code and test); agreeing on division of responsibilities and work; knowledge of other workers' past, current and likely future activities; and a shared, consistent work process [1, 5, 7].

Various academic prototypes as well as some commercially available products have been developed for software process management and support. Examples include SPADE-1 [1], ProcessWeaver [5] and Oz [7]. These systems suffer from a range of deficiencies. For example, many deploy centralised client-server architectures in which a central server caters for process modelling and a centralised process engine provides computerised process enactment support. Examples of such systems are Regatta [10] and SPADE [19]. Disadvantages of such an approach include reduced

robustness and performance and insufficient support for distributed work environments.

Although support for activities such as collaboration, coordination and communication is essential for today's dispersed work-teams and companies, another frequent shortcoming of process support systems to date is the limited support for cooperative activities. ProcessWeaver [5] and EPOS [12] are examples of systems with a limited range of coordination strategies. Other weaknesses are hard to use process modelling mechanisms and insufficient integration with third-party tools.

Most existing systems employ a homogeneous process engine, meaning that the process engine constitutes one single processing instance for enactment processing. Many process management and workflow tools also use their own custom data management approaches, typically centralised in one server component. The problem with such an approach is that all processing and data management is entrusted to one processing instance, reducing the robustness and flexibility of the engine. Some work has been done using de-centralised process engines to overcome these limitations, such as Serendipity-II and SwinDeW [9, 22]. While these tools provide decentralised and peer-to-peer workflow engines, they use single custom components for their process modelling and enactment.

Many workflow tools have used distributed object technologies such as CORBA [7, 15]. However many of these solutions provide limited dynamic extension and discovery of resources due to the limitation of these object technologies. Recently there has been interest in developing solutions that use web services infrastructures to enhance run-time discovery of workflow nodes and support easier integration of third-party tools [4, 22]. While these provide improvements in the scalability and reliability of the workflow system architecture, very few approaches to date take a service-oriented view of the workflow components themselves, allowing a variety of workflow engine services to be modularised and discovered at run-time [22]. Business process modelling and integration using web services has become popular, and initial approaches extending this for workflow support have been developed [16, 4, 20]. Our aim in this work was to explore whether process technology could be developed employing a set of services rather than one monolithic server component as in traditional process management and workflow tools. Thus, we wanted to experiment with a service-oriented and highly distributed architecture for workflow tools to see if we could address some of the deficiencies of existing tools outlined above. The requirements for the process tool (IMÅL) we wished to develop were that it:

- Has a configurable process modelling notation, supporting end user tailoring of the notation.
- Supports modelling and enacting simple process flow, but with support for much more complex rules which are sometimes needed.
- Uses a decentralised process management engine using a set of services, allowing deployment of the system across multiple machines for fault tolerance and load balancing. The ability to upgrade process services while the system is in use is highly desirable.
- Reuses 3rd party Commercial Off The Shelf (COTS) services where possible as individual service components in the process engine. These should be interchangeable with other compatible services.

3 The IMÅL Approach

The approach we adopted for IMÅL focuses on a highly distributed and service-oriented process enactment engine. We use an existing, user-tailorable modelling tool to provide process modelling and visualization functionality, and user interaction with the system. An existing 3rd party rule-based engine component (Idiom) and document display tool (Microsoft InfoPath) were used to provide complex task automation and document management respectively. A prototype of IMÅL was built in order to investigate the advantages associated with such an approach. IMÅL's architecture is shown in Figure 1.

A meta-CASE tool, Pounamu [23], was used to define a visual process modelling notation and enacted process visualisation support tool, providing the main point of user interaction with the system. Process models are uploaded to a process model service that provides persistent process definition management. Models are run by one or more process engine services, two prototypes of which we have developed. One includes a simple flow-based engine and the second a complex rule-based engine. These enact parts of the process model and interact to coordinate their operation. A process state service provides persistent state information management for all enacted processes. A notification service detects changes to process model, state, resources (documents) and agents (users and automated services), informing subscribers of these events to support reactive process management. A web based process state service, user login service and document editing service interact with the other services to provide a range of user interface and user profile management functionality.

Third-party tools are integrated with the modelling tool and engine using a service-oriented architecture. Pluggable enactment engines provide functionality to interpret the process models and allow for computerised process enactment. The process engine was split and distributed as several smaller web services with individual responsibilities, including a simple event based processing instance. Information about process state and allocated tasks is made available via a web based to-do list service.

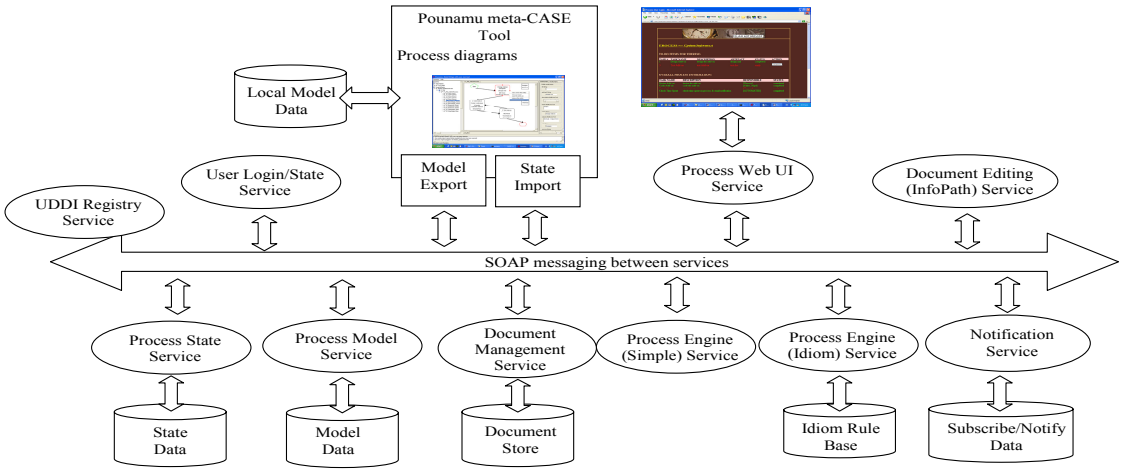

Figure 1. Decentralised architecture of the IMÅL process tool.

All of these services may run on different host machines. For example all modelling tools are installed and run as separate services on each user's PC. All services interact via SOAP XML messages and services may be discovered and integrated into the architecture at run time using a UDDI web services registry look-up. Many of these services use translation to and from different SOAP messaging protocols via the XSLT transformation scripting language. For example, the process model managed by the Pounamu modelling environment is translated to and from the format used by the process model service to read and write the model state managed by this service.

4 Process Modelling Language

Our IMÅL prototype supports a Serendipity-style [10] visual process modelling language, but provides a new and unique implementation of the interpretation and execution of these process models. The process modelling and visualisation language can be tailored by the user in limited ways using the meta-CASE tool used to realise the process modelling.

A visual meta-CASE tool, called Pounamu [23], was used to create a simple visual process modelling language and tool for IMÅL. It was also used to provide one of the main user interaction points for enacting and monitoring IMÅL processes. The Pounamu Process Modelling Language (PPML) was created and used when developing the Pounamu Process Modelling Tool (PPMT). PPML is a simple visual language based on that of the Serendipity process management tool [10]. PPML is somewhat less expressive that Serendipity's but offers sufficient modelling capabilities for the purpose of providing computerised support for most kinds of process enactment. The language can be enriched with further modelling constructs using the meta-tool and more complex enactment rules using the 3rd party Idiom rule-based engine.

Table 1 contains a summary of the modelling elements contained in PPML. PPML supports the concept of unique stage identifiers and allows for representation of the roles, artefacts and tools involved in process work. Additionally, PPML allows users to specify different input and output states for stages, which helps guide process flow when the process is being enacted. Another essential feature of PPML is the association of a service with stages. The service associated with a stage defines which processing instance is responsible for processing the stage's work activities. If no service is specified, a default processing instance is used. Finally, PPML offers a separate model element with which to model role-actor associations.

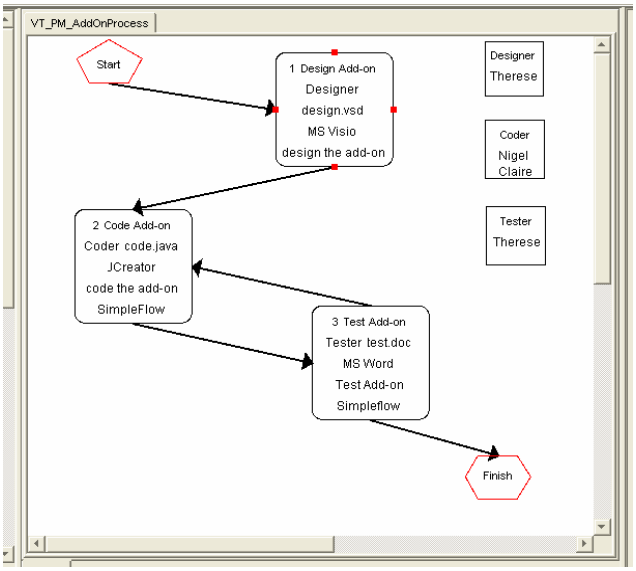

Figure 2. A simple IMÅL software process modelled using PPML and PPMT.

Table 1. Core elements of the IMÅL Process Modelling Language.

Notational Symbol	Example	Description	Notational Symbol	Example	Description
ID name description role artefact tool service inputs outputs	3 Testing test system Tester test.doc MS Word simpleflow start test succeeded failed	**BaseStage** – a basic stage in a process that involves work to be completed. It has a unique ID, name, a description of work involved, the role responsible for the work, the artefact and tool involved, and service responsible for processing the stage.	operator	OR	**LinkStage** – a join of two or more basic stages which flow into one other basic stage. Can contain the operators AND, OR or XOR. An OR operation can also be modelled by several connectors into one stage. More complex expressions can be defined which are interpreted by the 3rd party Idiom decision suite software component.
name	start	**StartStage** – the first stage in a process. When a process is started, a flow from a start stage into a basic stage starts the process enactment.	conditi on	if(1 > 0) else	**ChoiceStage** – a split of the flow from one basic stage into flows to two or more basic stages depending on a condition. It contains the value of the condition.
		Flow – a representation of process flow. Flows drive the process according to the output state of the stage it starts from and the input state of the stage(s) it ends in.	role actors	Tester John Therese	**RoleActor** – an association between a role and the actors that fill this role. It contains the name of the role as well as a list of actors filling the role.
name	finished	**StopStage** – the last stage in a process. When a flow reaches this the process ends.	Resource Type	Design Visio doc	**Resource** – a resorce used by a process stage/role actor associated with a process stage.

Figure 2 shows the result of using IMÅL to model a simple software process. This process describes basic steps in the update of an existing software program by adding new functionality to it. In this model, three stages are defined along with associated resources and flow of work between stages, as well as role associations for three users.

5 Example Usage

We briefly illustrate usage of the IMÅL process management system with an example software process.

5.1 The Pounamu Process Modelling Tool

Figure 3 shows a screen shot of a process being modelled with PPMT. Start and stop stages, three base stages and five flows have already been added to the view, and the user is about to add a role-actor association. Once the user has finished adding role-actor associations, the view contains a simple process model and a process engine can be used to enact it. The modeller also supports multiple model views for each process model allowing complex process specifications to be broken into related pieces across several diagrams.

In this example the process modeller has described a simple design/code/test/accept process with a loop back to the coding process stage from the testing stage. The finishing condition for the testing stage is then specified which causes the loop back. Various characteristics of each stage including role, resources and how the stage is enacted are also specified. For example, each stage in this example uses the SimpleFlow process engine which implements a simple workflow enactment.

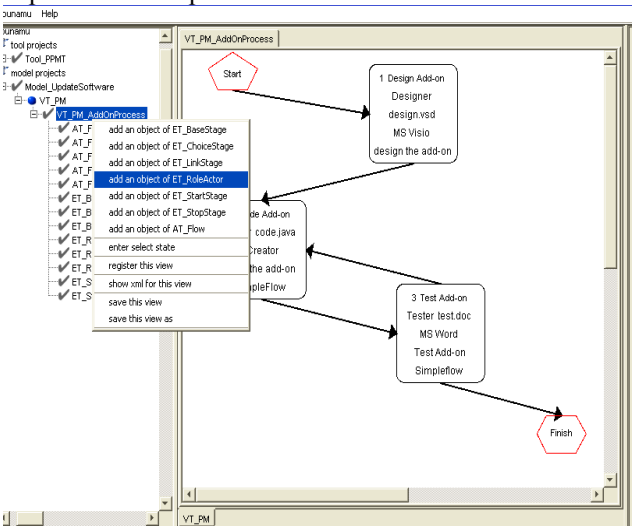

Figure 3. Process modelling in IMÅL.

5.2 *Enacting a Process*

In order for process enactment functionality to be enabled, a process engine must be initialised and the user must start the process from the start stage drop down menu. When this is done, the first stage(s) in the process model turns red to indicate that it is ready to be enacted. This scenario, with a user about to enact the first stage, is depicted in Figure 4. A stage that is currently enacted has the colour blue, whereas completed stages turn green. Once a process has been enacted, the user can make use of the to-do list service to view ongoing information about the process. By plugging in a new instance of the engine for each process model, IMÅL allows for several processes to be enacted concurrently.

The enacted process state changes each time a stage completes, is suspended or terminates. The user can directly indicate these state changes via a pop-up menu associated with each stage in the Pounamu process modelling tool. Alternatively they can interact with a web-based to-do list. When completing a process stage the IMÅL engine determines the next stage(s) to enact based on the process model specification and enactment engine associated with the process stage. In our current prototype most stages use the SimpleFlow engine while some with complex rules use the Idiom rules engine.

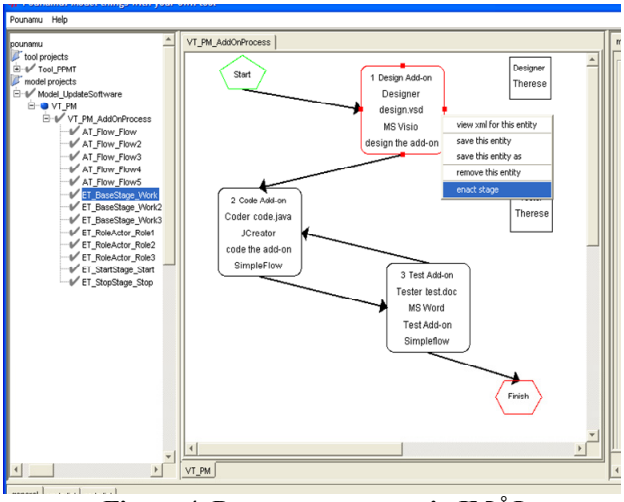

Figure 4. Process enactment in IMÅL.

5.3 *The Web-based To-Do List*

The to-do list is a key part of IMÅL, as it provides users with visual process state and progress information as well as functionality to directly enact processes. Users may use the to-do list to monitor and advance process state, without necessarily needing the heavy-weight Pounamu-implemented PPMT modelling tool. In order to get access to the web based to-do list, users point their browser to the address of the service, enter username and password and select the login button. The username and password are checked against a database of users, and if the login is valid, users can click a view to-do list button to view information about their responsibilities in the process as well as overall process information. When a user requests information by clicking the view to-do list button, the information retrieved is displayed in the browser. Our to-do list shares the same colour scheme used to show process state in the process model in Pounamu. Furthermore, it also includes functionality to enact the process model directly from the service. Figure 5 shows an example to-do list for a simple software update process. The to-do list may be configured by the user to refresh the display in the web browser periodically. This provides the user with an up-to-date overview of the current process state and other user activities.

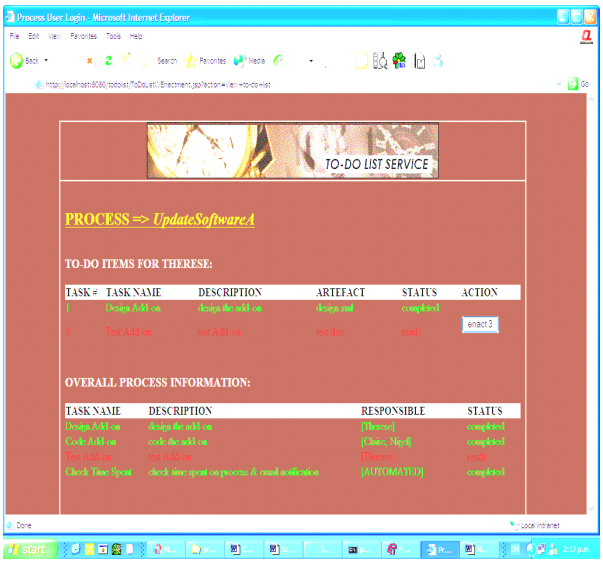

Figure 5. The IMÅL web-based to-do list.

5.4 *Process Automation with InfoPath and Idiom*

IMÅL provides users with the ability to view and edit process-related documents and to specify complex decision points in a process. Two existing tools, MS Infopath and Idiom Decision Suite, were integrated into IMÅL to provide these features. Infopath is an XML based tool in the Microsoft Office suite, which enables teams to easily gather and share information using rich, dynamic forms [14]. An InfoPath server is used to provide data input and output via a browser for enacted IMÅL processes. For example, Figure 6 (a) shows InfoPath being used to display time information for the simple maintenance software process. The XML document has been generated by the workflow engine and InfoPath invoked to display this using an InfoPath form. The user may update the time

estimate in the InfoPath form, which is then extracted for use by the process model's time monitoring sub-process.

Idiom is a tool that captures and deploys rule-based business decisions [11]. Rather than build a complex, custom rules engine for IMÅL we chose to implement a simple process flow engine and allow users to specify complex rules using Idiom. Thus IMÅL uses idiom to provide complex rule-based processing logic for stages, and also to automate activities based on rule-based decision logic. Idiom was used in the example software process scenario to track and process total time spent by members of a team on a process, and to trigger actions depending on the result of its rules processing. When the enactment engine receives an automatic stage for processing, it delegates it to the Idiom rules automation service. The automation service uses previously specified Idiom rules to calculate the total time spent by the team so far, compares it against an estimated time for the process, and automatically sends an email notification to the process manager if the time spent is more than 10% over the estimate. Figure 6 (b) shows an example of an automatic email notification that has been sent by Idiom as a result of processing of an IMÅL automatic stage.

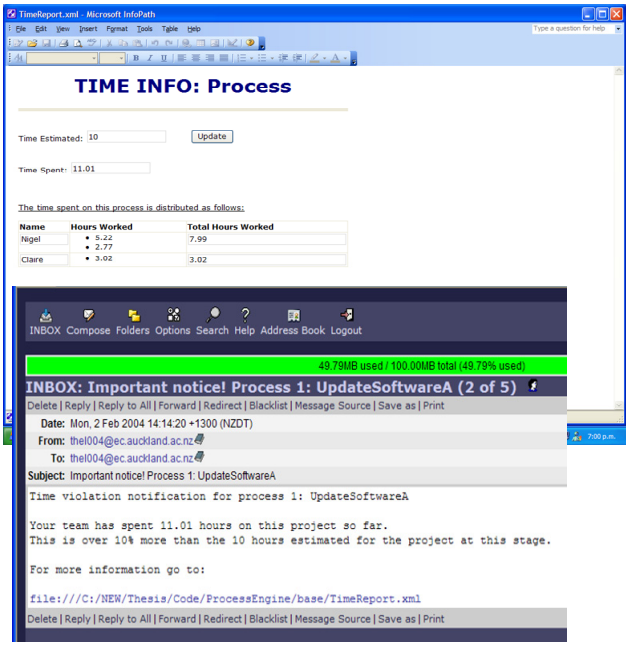

Figure 6. (a) MS InfoPath displaying a document; and (b) an Idiom-generated automatic Email notification.

6 IMÅL Design and Implementation

In this section we present the design of the IMÅL system with an emphasis on the underlying service-oriented architecture employed by our prototype.

6.1 Service-Oriented Architecture

When developing IMÅL we wanted to experiment with the concept of a set of co-operating services to realise a complex process modelling and enactment tool with 3rd party component integration via services. Depicted in Figure 7 is an outline of the service-oriented architecture that we have employed for IMÅL. The architecture is based on a collection of web services with different responsibilities that work together to support process modelling, enactment and third-party tool integration. The front end of the system is the user interface, which is provided either by the process modelling tool or a web browser. Further, the user interface is linked to underlying system information stored in various repositories via a web services layer. The process engine itself consists of a collection of web services with different responsibilities.

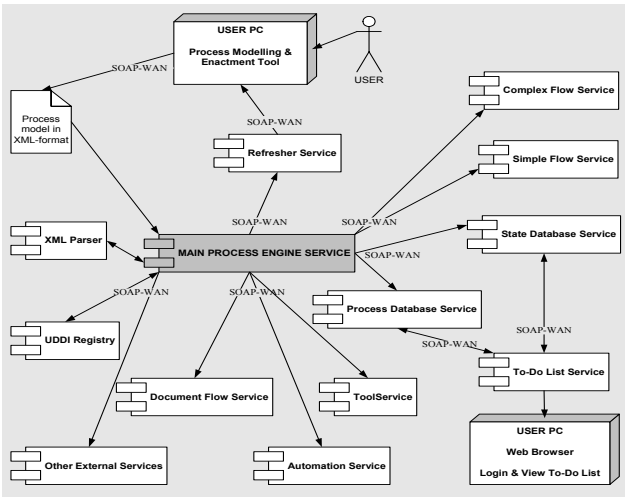

Figure 7. Architectural Design of IMÅL.

The following are key components and services that make up our service-oriented, distributed process engine:

- *Process Modelling & Enactment Tool* – responsible for user interaction with the system by allowing for modelling and enactment of processes. Additionally, this component provides a way to connect to and forward the process definition and enactment events to the main process engine service.
- *Main Process Engine Service* – coordinates process enactment using several services. The service uses the *XML Parser* for interpretation of process model information.
- *State Database Service* – stores information about stage state.
- *Process Database Service* – stores other process related information.
- *Simple Flow Service* – enables simple process flow by processing enactment events.

- *Complex Flow Service* – enables complex process flow for enactment events.
- *Refresher Service* – stores updates to the user interface and process model resulting from the processing of enactment events.
- *To-Do List Service* – displays to-do lists for users that log in to the service.
- *Tool Service* – presents the user with third-party tools needed to perform work.
- *Automation Service* – provides automation of tasks that need no user intervention.
- *Document Flow Service* – controls the flow of documents and artifacts for enactment.
- *Other External Services* – includes additional external services that the main process engine might beneficially make use of, such as awareness and collaboration services.
- *UDDI Registry* – provides an interface for looking up and discovering services.

6.2 Component Interaction

The way in which the various IMÅL components interact is essential to the quality of the support for processes enacted by the system. Figure 8 (a) shows a sequence diagram of event flows between different system components in a process enactment situation (Figure 3 (b) shows the user perception of this scenario).

Process enactment is initiated by the user enacting a stage in a pre-defined process model in Pounamu (1). Because the process engine component subscribed to process events when it was initialised and plugged into the IMÅL architecture, it is notified about the enactment event when it occurs (2). The component forwards the event to the process engine service (3), which further delegates it to the simple flow service for processing (4). During processing, the simple flow service updates the state database (5) and stores necessary updates to the process model and user interface in the refresher service (6-7). Finally, Pounamu's process engine component retrieves the updates from the refresher service via the process engine service (8-9), and applies them in Pounamu (10-11). Figure 8 (b) shows important event flows between components in a view to-do list scenario.

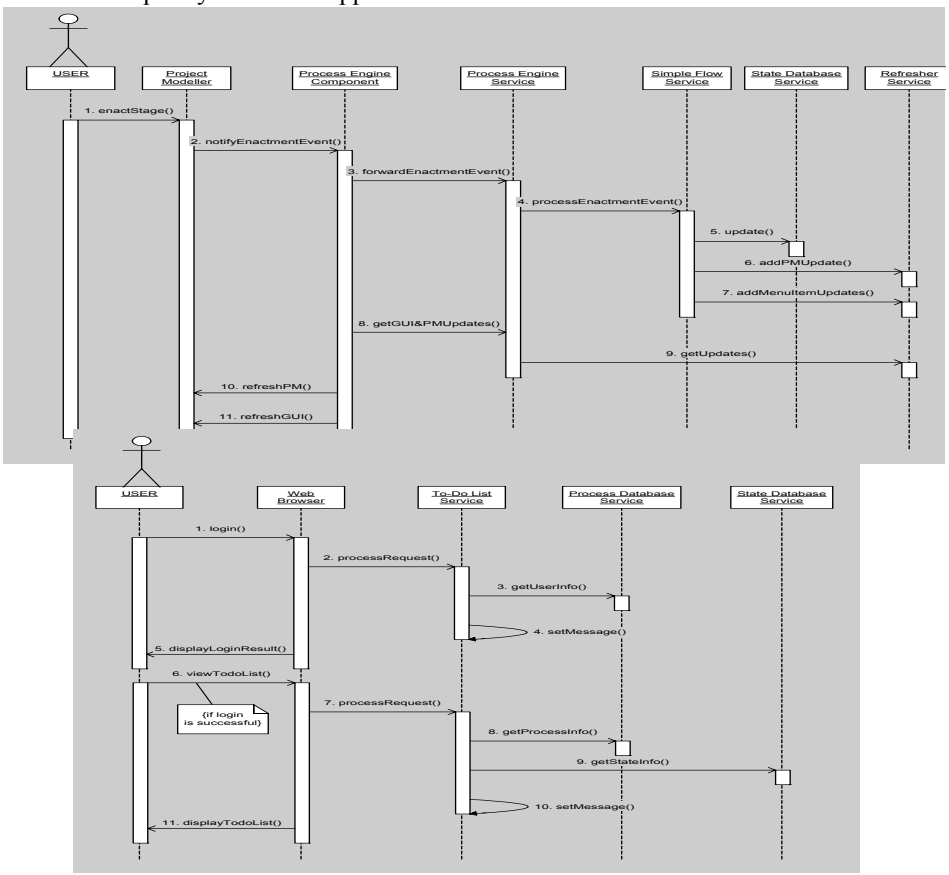

Figure 8. Sequence Diagrams of (a) enacting a Process Model and (b) to-do lists

The view to-do list scenario starts with a user pointing a Web browser to the address of the to-do list service. To be able to use the service, the user must supply his/her username and password and hit the login button (1). The login request is then processed by the to-do list service (2) by checking the username and password against a table of users in the process database (3). A message is set to indicate whether or not the login was successful (4), and the result of the login is displayed to the user (5). If the login is accepted, the user can choose a view to-do list option (6). Again, the request is processed by the to-do list service (7). The state and process database services are used to retrieve state and process information respectively (8-9). Finally, a message is set to indicate that the to-do list has been prepared (10), and the web browser displays the list to users (11).

6.3 Implementation

We used an existing meta-CASE tool, Pounamu, to define a visual process modelling tool for IMÅL. Firstly, the simple visual PPML, described in Section 4, was developed. Our PPMT tool was then constructed using Pounamu's meta-tool capabilities to realise the PPML. Pounamu enabled us to define a process modelling component for IMÅL that supports multiple views of software processes, different view types (process flow, resources, roles, document management), persistency of models, and distributed, collaborative process modelling. The PPML process modelling notation implemented by the PPMT tool can be modified by users via the meta-tool to change the look and feel of the process modelling diagrams while it is in use.

Pounamu utilises XML as the saving and loading format for all model projects. It was therefore natural to take advantage of this feature when building the process system. Hence, process models modelled using the PPMT are stored as XML automatically by Pounamu, and this XML save format is used to provide an interchange format between IMÅL's PPMT modelling tool and its process state management and enactment services. Pounamu provides a full web service-based API allowing it to be integrated as the process modelling component of the system as well as providing enacted process visualisation.

Web services are software technologies that support interoperable machine-to-machine interaction over a network [21, 19]. We chose web services as the technology with which to implement the service-oriented process engine of IMÅL. Advantages of web services include language and platform independency and good compatibility with commonly available XML management tools and standards. The IMÅL process enactment engine is built up of a collection of web services. A main service is responsible for coordination between all the smaller components of the enactment engine. Each service can run on a separate machine and be replicated, producing a highly distributed, scalable and robust environment. However, though the components are separate distributed web services, instances of each are all needed in order for the process engine to work properly as a whole. Communication between the services is facilitated by SOAP messages.

Services included in the current version of our IMÅL prototype are the main process engine service, state and process database services, simple flow service, refresher service, tool service, automation service and to-do list service. They were all implemented using Java and the Java Web Services Developer Pack (JWSDP). We used relational databases (SQL Server) as the storage facilities for the process engine state and user management services. These could be replaced straightforwardly by services using e.g. JDO, XML or object database technology, or LDAP server for user authentication.

We chose the Java Server Pages (JSP) technology to implement the to-do list as it is a thin-client technology providing platform independence and dynamic web pages. Our to-do list JSP embeds Java processing code within HTML code and accesses data from server-side components. When the page is displayed in a web browser, both the static presentation code and the dynamic content are displayed. Figure 9 shows how data access is achieved from the JSP page.

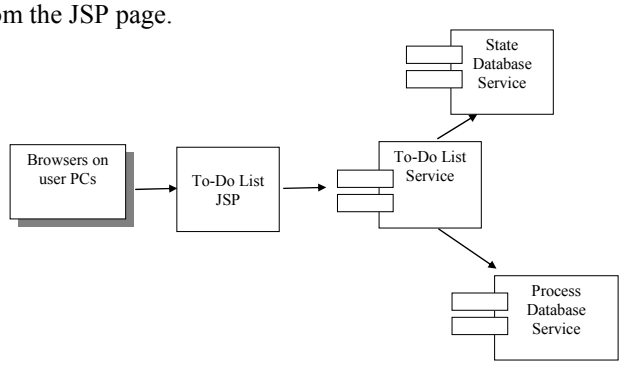

Figure 9. To-do list JSP and process state services.

MS InfoPath was used as our user interface realisation tool as it supports XML display and capture via web forms and a web service API. InfoPath was integrated via a tool initialisation service, which automatically starts up tools the users need to perform work on stages. The InfoPath instance is used to display pre-existing XML documents as web forms for users to view and modify, with the IMÅL process engine extracting document content from the XML for use in decision-making for the workflow. XML documents can also be generated by the workflow engine from process state and other document information and InfoPath forms are used to display this information to users.

The Idiom rule engine was integrated into the IMÅL architecture to provide a complex rules engine service.

IMÅL passes Idiom parameters from the IMÅL state and process database services and the Idiom rule engine service uses its rule base to process these. Results are returned to IMÅL for use in process decision making and display of information via InfoPath forms. Idiom may also invoke other third-party tools as a result of rule processing, such as email, messaging, spreadsheet or databases.

7 Evaluation

We have carried out three evaluations of our prototype IMÅL process management tool. One involved a group of users designing and enacting simple software processes collaboratively with IMÅL and completing a questionnaire with a set of closed and open responses. The second involved a Cognitive Dimensions assessment IMÅL's visual process modelling language to identify its strengths and weaknesses. The third was a performance evaluation of IMÅL under heavy loading.

A group of eight people evaluated IMÅL by carrying out a set of process modelling and enactment tasks relating to maintenance of a software system. Half of the participants were experienced with process modelling tools and half had used the Pounamu meta-tool's user interface before. The participants firstly reviewed and made small changes to a maintenance-oriented software process model. Some changes were made individually, some collaboratively via IMÅL with other users. They then enacted a process model and used this to coordinate work making small changes to an open source software application. Coordination activities included check-in and check-out of designs and code, email messages informing users of document changes and test results, and final agreement on completion of the process tasks. The survey demonstrated that our proof-of-concept IMÅL prototype using a service-oriented architecture and the functionality it offers were regarded favourably by users. Particularly, the feedback on the usability of the system was positive. There were no perceived performance or reliability problems with the tool, though only a small number of people were concurrently using it in our experiments.

Problems with our prototype included limited process enactment support via the "simple flow" engine, necessitating use of the Idiom rules engine design tool for more complex process flow decisions. Tailoring the process modelling notation requires use of the Pounamu meta-tool and this currently provides limited control over "safe" modifications for end users. It turned out to be easy for end users to break the process modelling tool when making changes via the meta-tool.

To follow up our user survey a cognitive dimensions evaluation was undertaken of the IMÅL modelling language. The Cognitive Dimensions (CD) framework was first introduced by Thomas Green in 1989 [8]. It provides a set of discussion tools for evaluating notations and information artefacts. We examined 14 characteristics of the modelling language including its appropriateness of mapping to the process modelling domain, support for multiple views and representation of dependencies, the tool's support for changing models, expressiveness of the notation and error-correction support in the tool. The IMÅL tool beneficially provides good closeness of mapping, high consistency and few hard mental operations for users. However the modelling functionality of the tool is reasonably error-prone and there is currently insufficient visibility and juxtaposability of models. Models are not viscous, i.e. it is easy to change them but the tool currently provides very limited validation of a model before enactment by the engine.

One concern often expressed with web services technologies and service-oriented architectures is their performance under heavy loading. IMÅL uses a comparatively large number of independent components, each modelled and realised as a web service, that are often combined into a single process engine instance in most other process management tools [22, 9]. For example, model representation, enacted model state, to-do list and user logon information are all potentially distributed services in IMÅL. Our current implementation of these services requires web service invocations between modules even if they are deployed on the same machine. We carried out a performance test of IMÅL distributing these services over multiple machines on a local area network, then ran a large number of process enactment events and assessed the number of operations supported by our distributed, service-oriented architecture. Our IMÅL prototype will comfortably support over 100 concurrent users assuming one enactment event per second per user, with associated communication between the various process enactment services. Response time per user from the to-do list and Pounamu visual process modelling tool is well under a second in this scenario. However, automated tasks in the process model will significantly reduce throughput of the enactment engine if inter-service communication is required. For example, complex rule evaluation by the Idiom rules engine, form rendering by InfoPath and database or CVS operations may adversely impact other IMÅL services in unpredictable ways, depending on their deployment scenario.

Several areas for future improvement of IMÅL exist. We currently do not have replicated instances of each service to support fault-tolerance and load-balancing for scalability. These are straightforward to add, but further performance analysis of the architecture is required to identify bottlenecks in communication and processing for each service, to identify correct replication and optimal deployment of service instances. We have to date only implemented one version of each engine service type (simple and Idiom-based automatic) one notification service interface (email) and one existing third-party

product integration (InfoPath). Adding alternative services that provide the same interface but different implementation would enable us to better investigate IMÅL's service generality and further improve this. Using other existing services e.g. user authentication, event notification services, and version control like CVS, would further demonstrate the benefits of the service-oriented architecture we have employed.

8 Summary

We have developed the IMÅL service-oriented process support environment. This offers functionality to model work processes as well as computerised support for enactment of the resulting process models. In order to enable process modelling and user interaction with the process system, the Pounamu meta-CASE tool was used to build a domain-specific process modelling environment. Further, a distributed and service-oriented process enactment engine was implemented as a collection of web services. This engine can be used to instantiate a process model generated by Pounamu at runtime, and provides the user with ongoing process enactment support throughout the process. Additionally, users can access information about the process state and tasks they are responsible for via a web based to-do list service. Task automation is supported by "automatic" stages whose decision logic is implemented by the Idiom rules engine acting as a service. Document management is provided by the Microsoft InfoPath tool as a web service. Evaluation of the IMÅL prototype has shown its process modelling and enactment support are usable and that its performance is acceptable for a moderately large number of concurrent users. Various potential enhancements include service replication for scalability and fault-tolerance, further third-party service integration, and supporting multiple, alternate services from different vendors for the same functionality.

References

1. Bandinelli, S.C., Di Nitto, D. and Fuggetta, A., Supporting Cooperation in the SPADE-1 Environment, IEEE Transactions of Soft. Eng., vol. 22, pp. 841-865, 1996.
2. Bandinelli, S. C., Fuggetta, A. and Ghezzi, C. Software Process Model Evolution in the SPADE Environment, IEEE Transactions of Soft. Eng., vol. 19, pp. 1128-1144, 1993.
3. Ben-Shaul, Z. and Kaiser, G.E. A Paradigm for Decentralized Process Modelling and its Realization in the Oz Environment, 16th Int. Conf. Soft. Eng., Sorrento - Italy, 1994, IEEE.
4. Cardoso, J. Sheth, A. Semantic Web processes: semantics enabled annotation, discovery, composition and orchestration of Web scale processes, In Proceedings of the 4th Int. Conf. on Web Information Systems Engineering, Rome, Italy, 10-12 Dec 2003, IEEE.
5. Fakasa, G.J., Karakostas, B. A peer to peer architecture for dynamic workflow management, Information & Software Technology, vol. 46, no. 6, May 2004, Elsevier, pp.423-31.
6. Fernström, C., Process Weaver: Adding Process Support to Unix, presented at 2nd Int. Conf. on the Software Process, Berlin - Germany, 1993.
7. Filho, R., Wainer, J., Madeira, E., Ellis, C. CORBA based architecture for large scale workflow, In Proceedings of the Fourth Int. Sym. on Autonomous Decentralized Systems, Tokyo, Japan, March 20 - 23, 1999, IEEE CS Press.
8. Green, T. R. G. Cognitive Dimensions of Notations, in People and Computers V, A. Sutcliffe and L. Macaulay, Eds. Cambridge - UK: Cambridge University Press, 1989, pp. 443-460.
9. Grundy, J.C. Apperley, M.D., Hosking, J.G., and Mugridge, W.B., A Decentralized Architecture for Software Process Modelling and Enactment, Internet Computing, Vol. 2, 1998.
10. Grundy. J. C. and Hosking, J. G. , Serendipity: Integrated Environment Support for Process Modelling, Enactment and Work Coordination, Automated Soft. Eng., vol. 5, pp. 27-60, 1998.
11. Idiom Software Ltd, Idiom, 2001.
12. Jaccheri, M.L. Larsen, J.O. and Conradi, R. Software Process Modelling and Evolution in EPOS, presented at 4th Int. Conf. on Soft. Eng. and Knowledge Eng., Capri, Italy, 1992.
13. McPherson, S. JavaServer Pages: A Developer's Perspective, April 2000.
14. Microsoft, Microsoft Office Infopath Product Guide, March 10 2003.
15. Miller, J.A., Sheth, A.P, Kochut, K.J., Wang, X. CORBA-Based Run-Time Architectures for Workflow Management Systems, Journal of Database Management, vol. 7, no. 1, 1996, 16-27.
16. Shen, J., Yang, Y. Zhu, C., Wan, C. From BPEL4WS to OWL-S: Integrating E-Business Process Descriptions, 2005 IEEE Int. Conf. on Services Computing, Orlando, USA, July 2005.
17. Sun Microsystems, Developing XML Solutions with JavaServer Pages Technology, 2003.
18. Swenson, K.D., A Business Process Environment Supporting Collaborative Planning, Journal of Collaborative Computing, vol. 1, pp. 15-34, 1994.
19. Systinet Corporation, Introduction to Web Services, Cambridge, Massachussetts, USA 2003.
20. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M. Analysis of Web services composition languages: the case of BPEL4WS, In Proceedings of 22nd Int. Conf. on Conceptual Modelling, LNCS 2813, Springer-Verlag 2003, pp.200-215.
21. W3C, Web Services Glossary, August 2003.
22. Yan, J., Yang, Y., and Raikundalia, G.K., SwinDeW - A Peer-to-peer based Decentralised Workflow Management System, to appear in IEEE Trans. on Systems, Man and Cybernetics.
23. Zhu, N., Grundy, J.C. and Hosking, J.G. Constructing domain-specific design tools with a visual language meta-tool, CAiSE 2005 Forum, Portugul, June 2005, Springer.