Jan Martin Jansen, Pieter Koopman and Rinus Plasmeijer reprint

Web Based Dynamic Workflow Systems and Applications in the Military Domain

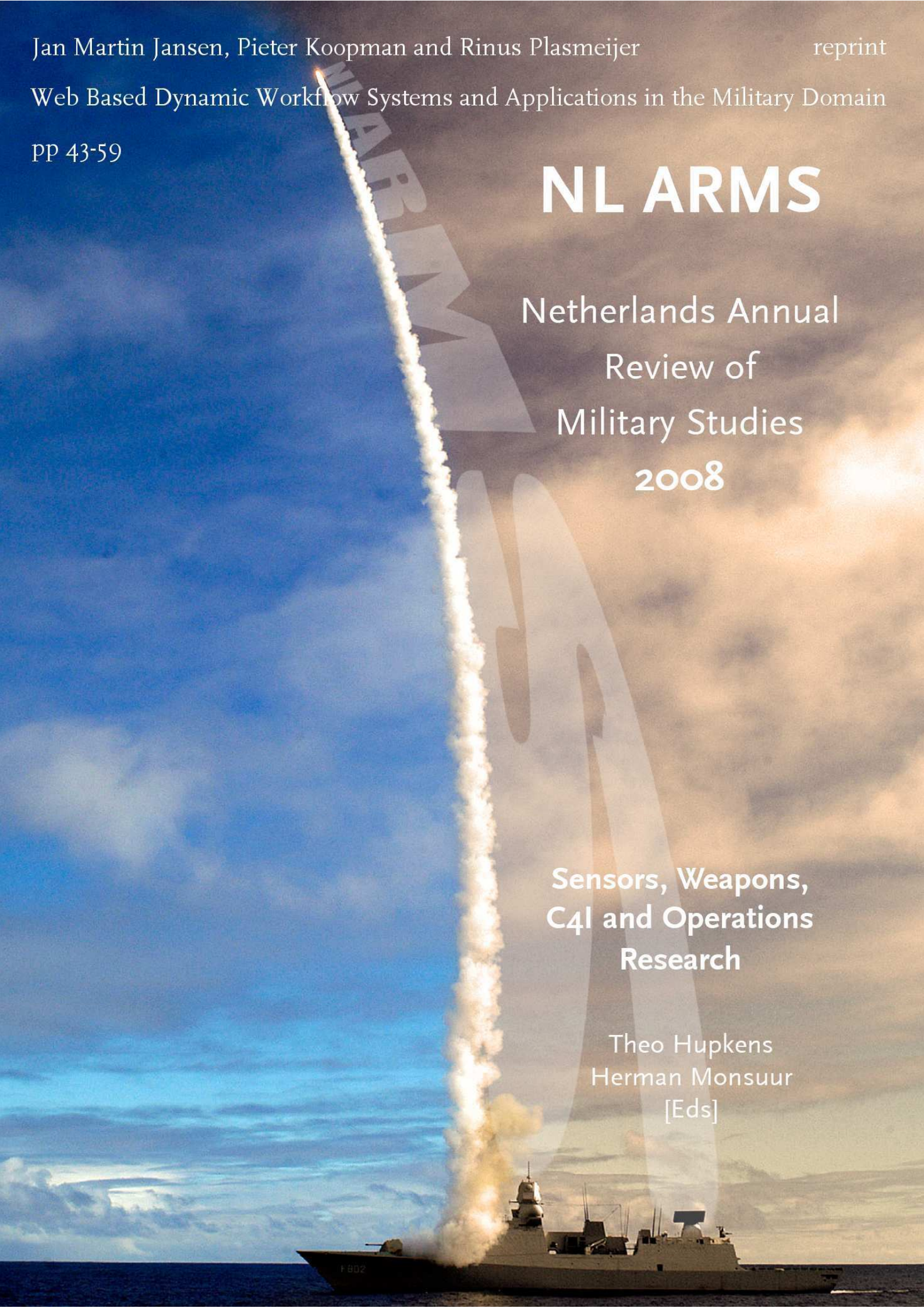
pp 43-59

NL ARMS

Netherlands Annual Review of Military Studies 2008

Sensors, Weapons, C4I and Operations Research

Theo Hupkens
Herman Monsuur
[Eds]



Web Based Dynamic Workflow Systems and Applications in the Military Domain

Jan Martin Jansen, Pieter Koopman & Rinus Plasmeijer**

Introduction

A workflow system is a computer system that guides and monitors the tasks that have to be done by human workers in collaboration with computers. The total amount of work is a structured collection of tasks. The order of tasks and the assignment of tasks to workers is specified in the work flow specification. Typical application areas are activities where information has to be shared and enriched or checked by several people and/or systems from different functions, disciplines, departments, or even organisations. Examples are: insurance claim handling, purchase ordering, distributed planning, and distributed execution of standard operating procedures. In fact, every activity that involves the transformation of information from one person to another or between persons and automated systems can be modelled as a workflow system. In military applications, workflow systems have the potential to speed up operational Command and Control and supporting processes like planning, logistics and administration.

Commercially available workflow systems mostly use a special purpose (graphical) formalism to specify tasks and the flow of information between tasks. From this graphical representation normally an application is generated.

In this paper we describe the iTasks workflow system [7]. The iTasks system is a software library written in the declarative programming language Clean, which allows for the high-level specification of multi-user workflows. An important advantage of the iTasks system above commercially available workflow systems is that it is not a special purpose system, but embedded in a general purpose programming language which allows the user a much higher degree of freedom to specify complex tasks and actions within these tasks. For example, in the iTasks system it is possible to specify workflows where new tasks are dynamically dependent or created on the results of previous tasks. The system also allows for easy and flexible encryption of the information exchanged between partners.

Another important advantage of the iTasks system is that its user interface is entirely web-based. This makes it possible to use the system in collaboration with external partners – perhaps widely distributed geographically – without the installation of special software at the partner's side.

Finally, the iTasks system allows for client side execution of (sub)tasks using a special purpose interpreter running in the browser at the client side. This minimises the use of bandwidth for exchanging information between client and server and guarantees a quick response. It is even possible to handle subtasks without having a connection to the server.

* Institute for Computing and Information Sciences (ICIS),
Radboud University Nijmegen, the Netherlands.

This interpreter is implemented as a Java Applet and can run in all current available web-browsers, again without the installation of special software.

Workflow Systems

Commercially available workflow systems mostly use a graphical tool to specify the workflow application. In the graphical representation nodes connected with dependency arrows occur. Some nodes represent tasks that have to be fulfilled by the user, other nodes specify control. Typical control nodes are used to enable the parallel execution of tasks or to synchronise the results of several tasks. Ref. [2] contains an overview of the most important workflow patterns.

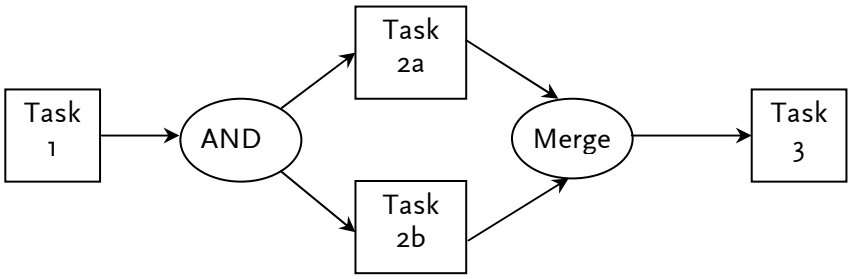


Figure 1. Example of a workflow specification

Fig. 1 shows an example of a workflow specification. Here the workflow consists of four tasks. Task 1 has to be executed first, e.g., the making of an initial plan. Tasks 2a and 2b can then be executed simultaneously, e.g., two subtasks that can be executed in parallel. After they both have completed, Task 3, e.g., the calculation of the total costs can be started. In this example the user tasks are represented by rectangular nodes and the control nodes by oval nodes. The specification only shows the flow of control and not the flow of information. The actual code for the workflow system is generated from the graphical representation by a code generator. The generated code mostly consists of a database program for storing information used in the workflow and several (web)applications for the users of the workflow. The applications interact with the database for retrieving information. In general, the database also contains the control information.

An important restriction of the use of a graphical tool is that the structure of the workflow is statically defined by the graphical tool and cannot dynamically change as a result of data produced by a preceding task.

The use of a code generator often leads to the creation of a large number of files, scripts, database tables and applications, where some of these must be further edited by the application programmer. This complicates the maintenance of the application and makes debugging in case of errors difficult.

Although workflow systems are useful for the support of all kind of activities in organizations, the actual use of these systems is limited. The main reason for this is that current commercial workflow systems do not accommodate the flexibility needed for practical use. They are only useful in situations where the actual flows can be formalised

beforehand in standard procedures. In practice more flexibility is needed because unexpected situations can occur that require ad hoc handling.

Dynamic Workflow systems, like iTasks, can be used to support processes with a dynamic nature. In this paper we look forward to the use of iTasks for planning and execution of military operations. These processes are characterised by a very dynamic nature. Planning processes need to be adapted because e.g., new intel information gives rise to changes. During operations e.g., feedback on the feasibility of plans or new sensor information may lead to adaptation of plans.

This paper gives an overview of the iTasks system, the motivation behind its implementation and looks forward to applications in the military domain. The structure of this paper is as follows. The paper starts with a justification of the use of web pages as interfaces for applications. Then the architecture of iTasks applications is discussed and an introduction to the iTasks library is given. The possible use of iTasks for the planning of complex military operations and the use for military operations themselves is discussed, together with a discussion of the use of iTasks in relation with Net-Centric Operations. Finally, the results are summarised and some conclusions are drawn.

Web interfaces

iTasks applications have a web-based user interface. In this section the advantages of web-based interfaces and the problems that arise from them are discussed. Using web pages as the interface for applications has gained much popularity during the last years. An important advantage of this approach is that no installation of special software is necessary on a computer to use the application. It is even possible to run the application on different platforms or operating systems (Microsoft Windows, Linux, Mac OS, Solaris, etc). Examples of applications with a web interface are: e-mail programs (web mail), online banking applications and web shops like Bol.com and Amazon.com. In fact, almost every (large) company nowadays uses web interfaces as the defacto standard to communicate with customers.

Despite this popularity and convenience for the user, for a developer of software it is still hard to write software for the web. The reason for this is that the web was originally developed to display information and links to other pieces of information. The current architecture of the web still reflects this original goal. An important problem that interactive web applications have to deal with is the fact that a user can move away from a web page at any moment and return to it later (e.g., by closing the browser or page, selecting another web page or by clicking the previous or next button). The web application must be able to deal with this. As a consequence of this transactions (e.g., the purchase of a book) are often not completed and the system should be capable of rolling back that part of the transaction that has already been completed.

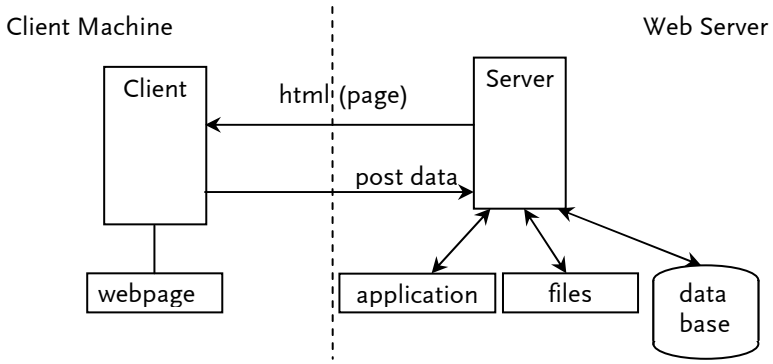


Figure 2. Architecture of a web application

Fig. 2 shows the typical architecture of a web application. The client browser (left from the dashed line) only displays the html generated by the (web) application running on the web server (right from the dashed line). This (web) application can read/write information from/to a database or files residing at the server side. The user can fill in web forms that are sent to the server and processed there. As a result the server produces a new web page that is displayed at the client side. The database and files are used to maintain information (e.g., the user login name or the purchase the user made). As already mentioned a difficulty that has to be dealt with is maintaining the state of a transaction. The server has to keep track of this state. The complication is that the user can move away from the web-page at any moment and come back to the web page at a later time.

In the classical setting the web server processes the web form filled in by the user and produces a complete new html page. A drawback of this approach is that the interaction can become rather slow. To overcome this, local processing at the client side can be done using JavaScript. JavaScript is a small programming language for which an interpreter is integrated in all modern web browsers. Web pages can be (partially) updated by JavaScript. In this way simple processing, that does not really need information available at the server, can be performed at the client side. To further enhance the performance of web applications it is even possible to make a request to the server from JavaScript where the results can be used to (partially) update the web page using so-called Ajax (Asynchronous JavaScript And XML) [3] technology. Fig. 3 shows the architecture of web applications using Ajax technology. Google extensively uses this technique in applications like Gmail and GoogleMaps to speed up their performance and make them more interactive. As a result of this, the developer of web applications has to write several programs: the server side program including access to the database and the generation of html pages; the JavaScript program for client side processing and Ajax interaction with the server. This makes software development of web applications a cumbersome and error prone activity. Again, like in the case of workflow software applications, there are tools that simplify this process. Most of these tools generate frameworks for web applications that must be filled in by the programmer. Again the developer has to deal with the problems of maintenance and debugging for these generated files and frameworks.

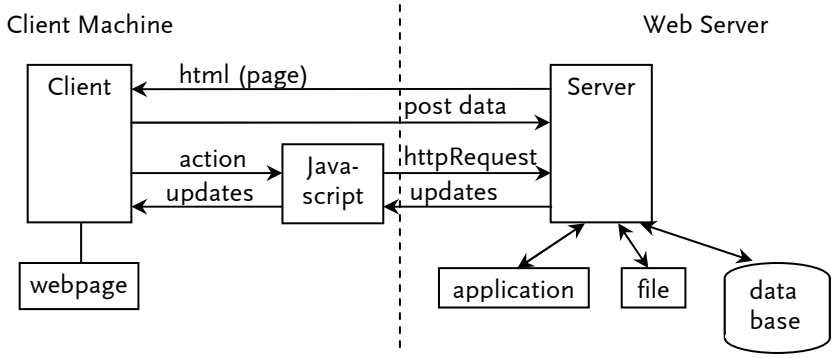


Figure 3. Architecture of a web application using Ajax technology

Architecture of an iTasks application

The iTasks library and iTasks applications are all programmed in the functional programming language Clean [8]. Clean is an example of a 'pure lazy' functional programming language. Another example of such a language is Haskell [1]. Important properties of pure lazy functional programming languages are:

- pure functional: a program consists of functions only and the result of each function is completely determined by the value it returns (there are no side effects);
- lazy: only the calculations necessary for obtaining the end result are done. This makes it possible to use infinite data structures like trees and lists without calculating them completely;
- memory (de)allocation is automatic;
- they have a powerful (strong) type system, which allows for the early detection of programming errors;
- they allow for higher order functions (functions that have other functions as arguments and result);
- they allow for generic functions (functions that can handle arbitrary data types).

The iTasks combinator library depends heavily on all these properties and can be seen as a major example of the application of generic programming techniques. For example, the generation of web forms from data structures, the (persistent) storage and retrieval of data in files or databases, the handling of user updated html forms, are all programmed using generic techniques. The consequence is that an application programmer gets this all for free and does not have to program any of these issues. The combinators of the iTasks library are all higher order functions. It is impossible to program a library like iTasks in a traditional programming language like C, C++ or Java in the same concise way. Parts (subtasks) of an iTasks application can be executed at the client side of the application by an interpreter. This is realised by giving the developer the possibility to annotate tasks in the program with the '`OnClient`' annotation.

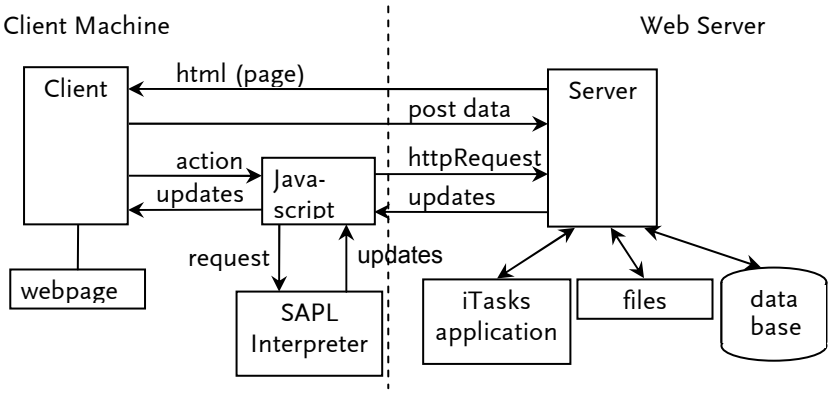


Figure 4. Architecture of an iTasks application

Fig. 4 shows the architecture of a typical iTasks application. This architecture greatly resembles the web architecture for Ajax applications, but there are important differences:

- At the client side the SAPL (Simple Application Programming Language) interpreter [4] is added. This interpreter is implemented as a Java Applet and is capable of executing Clean functions at the client side of the application;
- Both the server and client programs are generated from one single source programmed in Clean. From this source a server executable and a client SAPL program are generated by the Clean compiler. Both the executable and the SAPL source comprise the complete program. In theory it is even possible to run the complete application at the client, except for the storage and retrieval of information in files and data bases;
- All storage actions at the server side are automatically generated (at run time) from the data types in the program. No special user code is necessary, only the marking by the user of the data that have to be stored;
- The JavaScript at the client side is generic (the same for all iTasks programs). The JavaScript acts as an intermediary between client and server and client and SAPL interpreter. It takes care of updating the page with results from the server or from the interpreter and it transforms user actions in the forms into calls for the server or the interpreter;
- Web pages are generated within the application with the use of generic techniques, so the programmer does not have to program html pages;
- The developer only has to deal with the Clean source program. There is no need for editing generated program sources. This simplifies the maintenance of iTasks programs considerably.

As a consequence of this the programmer only has to deal with a single Clean program and not with JavaScript, html pages and data bases. This simplifies the creation of applications significantly. As will be shown, iTasks programs are very compact.

The Clean compiler produces both an executable and the input for the interpreter. The interpreter is a Java Applet, which is part of the initial html page and is loaded in the web browser when this page is loaded. After starting the interpreter the input file for the interpreter is loaded from the server by a JavaScript function.

Implementation aspects of iTasks

The iTasks library is built on top of the iData library [6]. iData is a library that supports the automatic generation of web pages. This library is based on the following two principles:

- Each user-defined data type can be turned into an (editable) web form using generic functions from a library included in iData. The web form is generated at the server side and transmitted to the browser as (part of) a web page;
- Each user edit action of such a web form is automatically transformed into an updated instance of a data type by a generic function. The user edit action is uploaded to the web server and further handled by the generic machinery. The data type of the value the user entered is checked automatically; if it is incorrect the update is not made.

The programmer may change the default generation of web forms by giving a specific instance of the above-mentioned generic functions for a data type. In this way custom-made web forms can be generated.

So, iData takes care of displaying (editable) information in a web form and updating the data structure in the application with changes made in these displayed web forms. A simple iData application consists of a single web page where a user can fill in forms. iTasks adds the following concepts to iData:

- Tasks are the basic units of an iTasks application. A task can consist of a single iData form (editable data type), but can also be a combination of simpler tasks (using combinators). The most important addition to an iData application is that the user can finish a task by clicking the 'Done' button that is added to a web page. At that moment the form corresponding to the data type of the task cannot be edited anymore and its content becomes available to other tasks;
- Task combinators enable the combination of tasks. A large number of basic combinators are available in the library, but the programmer can also define new combinators. Combinators are used to control the flow of processing and data from one task to another. Tasks can be performed sequentially, in parallel and distributed over several users.

An iTasks application starts as a single executable application at the server side. The iTasks application is re-executed for each client action and generates (part of) a new web page using as input the action and the current state. The state of an iTasks application can be encoded in the web page (in hidden fields), in server side files or in a server side database or a combination of these.

Not all tasks need to be executed by the server. Smaller tasks that do not need information stored in files or databases at the server side can be executed by the SAPL interpreter at the client side. The client side execution of tasks is completely transparent to the programmer of an iTasks application. Only the annotation of a task by the 'OnClient' keyword in the Clean source is necessary.

The client side execution of tasks increases the (execution) performance of iTasks application by reducing the internet traffic overhead (no client-server-client round trip necessary). The SAPL interpreter can also be used for the efficient implementation of (mouse) event processing for more interactive web elements like drawing canvases.

The iTasks combinator library

An iTasks application consists of a set of tasks (possibly for several users) that will be processed in the order specified in the workflow Clean program. iTasks allows for both sequential and parallel evaluation of tasks. The workflow evolves dynamically: the outcome of previous tasks determines the future behaviour of the workflow. Submission of a form by a worker will therefore generally influence the remaining work of this user as well as that of other workers: it may cause the creation of new tasks or the termination of existing tasks. Hence, a single click on a page in a browser may cause very complex state changes on the server and can affect the work of many workers.

The iTasks system is compositional: this means that complex tasks can be composed from other (smaller) tasks by using the iTasks combinators. The basic building blocks for tasks are editable data types. The iTasks system is capable of turning any data structure into an editor. The editor is displayed as a web form in a web page and can be edited by the user.

In iTasks it is also possible for the user to specify new combinators or parameterised workflows. Parameterised workflows can take other workflows as argument. Later on, an example of such a workflow will be given.

Below the most important iTasks combinators will be introduced by giving simple examples of their use. The examples serve to give the reader an idea of the potential and expressiveness of the iTasks system. With the iTasks toolkit it is possible to make much more complex workflows. Details can be found in [6].

Turning data types into tasks with editTask

The '`editTask`' function can turn an element of an arbitrary data type into a task. As a result the user can edit the data type element in a web form. The result of the edit action is fed back to the iTasks system as an element of the data type and can be further processed. If the type of the input does not fit the type of the data type the update is not made. '`editTask`' has two arguments: the name of the button that the user should press to end this task and the initial value of the editor. Here two examples of the use of this function are given: one for an integer argument and one for an element of type Person (together with the definition of Person).

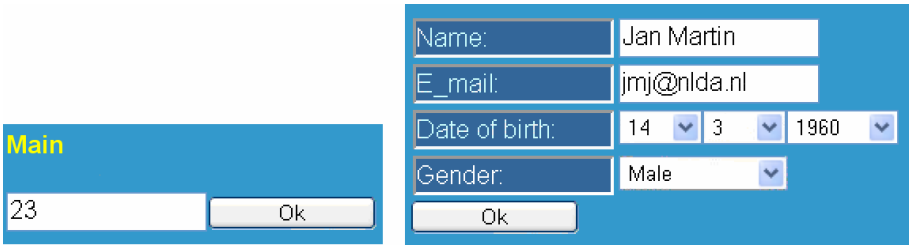


Figure 5. editTask for Int (left) and Person (right)

```
simpleInt :: Task Int
simpleInt = editTask "Ok" 0

:: Person =  { name        :: String
             , e_mail      :: String
             , dateOfBirth :: HtmlDate
             , gender      :: Gender
             }
:: Gender =  Female | Male

simplePerson :: Task Person
simplePerson = editTask "Ok" createDefault
```

Fig. 5 shows the resulting editors. Note that Person has 'createDefault' as initial value. The fields in the form will now get default values generated by the system.

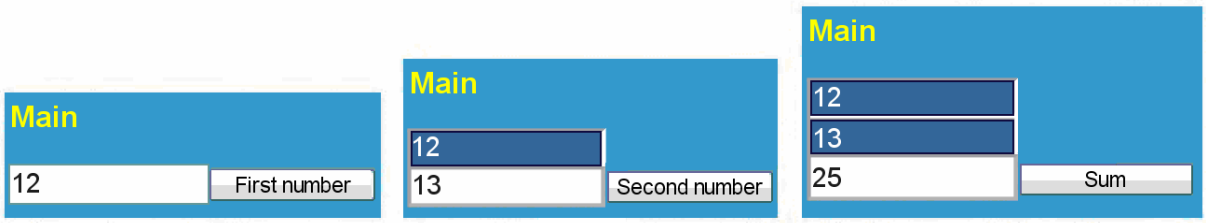


Figure 6. Sequence example (three consecutive windows from left to right)

Sequence

Workflows can be created by combining simple tasks where information of one task (e.g., the result of filling in a form by a user) flows to another task. The combinator in the iTasks system that takes care of this flow is the '=>>' operator. As an example consider the situation of a single user who has to provide two integer numbers in two consecutive forms and as a result gets the sum of these numbers.

```
simpleAdd :: Task Int
simpleAdd
=           editTask "First number" 0
  =>> \v -> editTask "Second number" 0
  =>> \w -> editTask "Sum" (v+w)
```

Fig. 6 shows the three consecutive appearing web-pages. '=>>' takes the result of the left side and gives it as an argument to the right side. '`\a -> body`' is a notation to introduce an inline anonymous function with argument '`a`' and body '`body`'. So '`=>> \v ->`' can be read as: make the result of the left hand side available to the right hand side under the name '`v`'.

Normally the final result of a task should not be editable. This can be achieved by replacing the last line by: '`=>> \w -> return_D (v+w)`'. Now the result is only displayed.

It as also possible to display a text next to the result:

```
=>> \w -> [Txt "Their sum is"] !>> return_D (v+w)
```

Sequence with multiple users

The previous example can easily be turned into an example with multiple users. Here user 0 first has to enter the first number, then user 1 should enter the second number and user 2 gets the result. The user number must be selected with a drop down box on the page, but can also be associated with a user login.

```
simpleAddMU :: Task Int
simpleAddMU
=           0  @:: editTask "First number" 0
  =>> \v -> 1  @:: editTask "Second number" 0
  =>> \w -> 2  @:: editTask "Sum" (v+w),
```

where '`k @:: t`' assigns task `t` to user `k`.

AND parallelism

The AND (`-&&-`) operator generates two tasks that both have to be finished before the results can be used.

```
simpleAndMU :: Task Int
simpleAndMU
=      (0  @:: editTask "Number entered" 0)
  -&&- (1  @:: editTask "Number entered" 0)
  =>> \(v,w) -> 2 @:: editTask "Sum" (v+w)
```

Now both user 0 and 1 may enter their numbers in parallel. The results are collected; their sum is calculated and displayed to user 2.

For AND also a multi-version '`andTasks`' exists, which handles a list of tasks. The task completes when all subtasks are completed.

OR parallelism

The OR (`-||-`) operator generates two tasks in parallel. As soon as one of them finishes the result of that task is available. The result of the other task is ignored.

```
simpleOrMU :: Task Int
simpleOrMU
=      (0   @:: editTask "A number" 0)
  -||- (1   @:: editTask "A number" 0)
  =>> \v -> 2 @:: editTask "First number" v
```

Both user 0 and 1 may enter a number, but only the first one that completes is received by user 2.

Also for OR a multi-version '`orTasks`' exists, which handles a list of tasks. The task completes as soon as one of the tasks completes.

With the help of sequence, AND and OR a great part of typical workflow applications can be programmed. The workflow literature often distinguishes between splitting and merging [2]. Splitting is the start of several tasks in parallel and merging is collecting the result(s) of one or more of the tasks that are split. In this way AND and OR both have the same splitting operation but have different merge operations. Because in iTasks the passing of control and results is integrated, splitting and merging are not distinguished.

Parallel tasks with a condition

In iTasks a special version of '`andTasks`' exists: '`andTasksCond`'. A number of tasks can be started in parallel. Each time one of the tasks is finished a condition is applied to all completed tasks. If the condition is met, '`andTasksCond`' is finished and the completed results are returned in a list.

```
simpleAndTaskCond :: Task Int
simpleAndTaskCond
=  andTasksCond pred [("User" +++ toString u,
                       u @:: editTask "Number entered" 0) \\ u <- [1..4]]
  =>> \xs -> [Txt "Their sum is"] !>> return_D (sum xs)
where pred xs = sum xs > 3
```

Here a parallel task for 4 users is started. They all have to enter a number. Here the condition checks if the sum of the already entered numbers is greater than 3. As soon as this is the case this task stops and the results are passed to another task where they are displayed.

This is a very powerful combinator because many other combinators can be expressed using it. For example the definitions of '`andTasks`' and '`orTasks`' can be given by:

```
andTasks xs = andTasksCond (\ys = length ys == length xs) xs

orTasks xs = andTasksCond (\ys = length ys == 1) xs
```

Tasks with check

Very often the data entered in a form must be checked before the user can proceed. For this a special version of '`editTask`', '`editTaskPred`' is added.

```
simpleCheck :: Task Int
simpleCheck
=             [Txt "Enter a positive number"] !>> editTaskPred 0 checknum
  =>> \num -> [Txt ("You entered: " +++ toString num)] !>> return_D num

checknum num = (num >= 0,[Txt "Number should be positive"])
```

The user should enter a positive number. If a negative number is entered, the user gets a warning and may enter a new number.

Cancelling a task

A frequently occurring action is that a user starts a task, but that he decides to cancel the task. For cancelling a 'Cancel' button should be added to the form and an appropriate action should be taken.

```
simpleCancel :: Task String
simpleCancel
=       (editTask "Number entered" 0 =>> \num -> return_V [num])
  -||-  (buttonTask "Cancel" (return_V []))
  =>>   handleResult
where handleResult []  = return_D "User cancelled"
      handleResult [n] = return_D ("User typed " +++ toString n)
```

Now together with the number field a 'Cancel' button is displayed as an OR task. The task that handles the result will determine the status of the result (empty list or list with one element). In case of an empty list the user has cancelled the task.

Parameterised tasks

It is also possible to specify tasks that have forms of any type as parameter. In this way abstract frameworks to be filled in later with a concrete type can be made. As an example consider the following task frame for double-checking a user form (note that '`a`' is the type of the parameter and '`val`' is its value).

```
doubleCheck  :: a (a -> (Bool, [BodyTag]))  -> (Task a) | iData a
doubleCheck  val pred
=                   [Txt "Please fill in the form:",Br,Br]
    ?>> editTaskPred val pred
    =>> \na ->    [Txt "Received information:", Br, Br,
                   toHtml na, Br, Txt "Is everything correct ?", Br]
    ?>> chooseTask      [("Yes", return_V True), ("No",  return_V False)]
    =>> \ok   ->  if ok (return_V na) (doubleCheck  na preda)
```

'`doubleCheck`' is an extension of '`editTaskPred`'. It first checks the value the user entered against the Boolean function '`pred`', and after this value is ok, it asks the user for a confirmation for this value.

Double check can for example be used to check a person form as follows:

```
doubleCheck createDefault personcheck
```

where '`personcheck`' is a Boolean valued function that checks a person type.

Shifting tasks

In the iTasks system it is possible to shift work from one user to another user. In the next example user 0 may start a task for user 1. This task consists of three steps. If user 1 completes the task the result is sent to user 0 and displayed. But if user 1 stops the task, the work is shifted to user 2 who has to complete the task.

```
shiftExample = simpleShift threeStepTask
```

```
threeStepTask :: Task Int
threeStepTask =   editTask "Done1" 0
      =>> \v1 ->  editTask "Done2" 0
      =>> \v2 ->  editTask "Done3" 0
      =>> \v3 ->  return_D (v1 + v2 + v3)

simpleShift :: (Task a) -> (Task a) | iData a
simpleShift task
=            button "Start the Work"
   #>> 1 @:: button "Stop" -!> task
   =>> \(stopped,TCl task) -> if (isJust stopped) (2 @:: task) (0 @:: task)
   =>> \result     -> return_D result
```

'`TCl task`' is a so-called task closure (partially evaluated task).

The possibility to shift partially evaluated tasks is very powerful and cannot be found in other workflow systems.

Other iTasks combinators

Up to now only a small number of simple examples are given. iTasks has other possibilities which will not be discussed in detail here.

- time limit for tasks, to put a deadline on a task;
- choose a number of tasks to complete from a list of parallel tasks;
- several kinds of repetitive tasks.

The task combinators from iTasks are not fixed, but can be easily extended with new combinators by an application programmer. This is possible because both the combinator library and the workflow program are written in the same language. In this way complex dependencies between data and tasks and even recursive tasks can be programmed.

Other uses of workflow formalisms

Workflow formalisms are not only useful to implement workflow systems, but can also be used to model procedures (or information flows) within a company or organisation. In this way a formal description of these flows or procedures can be made. The model can be used to check the completeness of the set of procedures. Are there steps in a procedure that do not have a follow up? Is the necessary input information available for all steps? The formal description can even be used for simulating the procedure and using this simulation to adapt the procedure.

Applications of iTasks in the military domain

The iTasks workflow library offers the possibility for the high level specification of complex workflows. This section will focus on the applications of iTasks in the military domain.

Workflows occur at various places within the military domain. First of all, standard workflow tooling for administrative processes like personal administration and travel expenses claim handling is used. Also for the material logistic processes, Enterprise

Resource Planning (ERP) tools are used. The Netherlands Ministry of Defence uses PeopleSoft, DIDO and SAP for these applications. In these areas the needs do not differ too much from the needs of other large companies and the commercially available tools are sufficient to deal with them.

Major operational processes are examples of more complex workflow problems. For these processes currently no workflow tools are used to support them, mostly because these processes are too complicated to fit in the formalisms of these tools. For operational processes one can distinguish between:

- the planning phase preceding a military operation;
- execution of the military operation.

Planning of military operations

The planning of operations comprises the following aspects:

- Logistics: Before people can be deployed, accommodation, power supply, water supply, food supply, etc. have to be arranged.
- Transport: Transportation is needed both for people and materiel (accommodation and supplies). A large part of the transportation has to be done beforehand (accommodation, fuel, infrastructure). Other transportation is needed during the entire deployment (food, ammunition, replacements).
- Intel: Prior to the deployment, but also during the operation, intelligence operations are needed. Examples of prior intel requirements are: What are the expected enemy forces, what is the available infrastructure (communication, resources (water, food etc))? What are safe routes for transportation? What are the local terrain conditions? How is the local climate? What kind of protection is needed for the initial transports? Examples of intel during the operations are: What is the enemy behaviour? What is the attitude of the local civilians?
- Communication: A communication infrastructure has to be built-up for the operation: radio, telephone (including GSM), satellite for communication with headquarters and allies including Non-Governmental Organisations, computer networks for the exchange of information, internet for home front communication.
- Budget: What will be the costs of the operation? Do we stay within the maximum allowed costs?

The planning process can be very dynamic, because, for example, intel information can influence already started other tasks. Planning of these complex operations often involves the commitment of large numbers of geographical distributed people over periods varying from several weeks to several months. Currently normal communication channels like telephone and e-mail are used for the exchange of information, while in general spreadsheet and database applications are used for maintaining information. This maintenance is in general on an individual or small departmental basis. This means that other people and departments do not have insight into this information and should make explicit requests (by telephone or e-mail) to obtain it.

It is clear that workflow tooling can be of great help during the planning phase of military operations. Here a summary of some issues the workflow system should support is given:

- access to information for the partners involved. We are dealing with a variable number of dislocated people causing a dynamic workflow topology;
- the automatic checking of deadlines and taking actions in case they are passed;
- initiation of actions (tasks) for several partners involved;
- monitoring the status of actions with the possibility to interrupt or reallocate tasks;
- the on-the-fly construction of new workflows by system end users;
- automatic checking of budget.

Use of workflows during operations

For the planning phase of military operations it is obvious that workflow tooling can be of great help. For the execution phase of military operations this is less clear. But many activities during the execution phase can be modelled as workflows. Military commands have to be distributed and refined. Feedback on the feasibility of these commands must be given. But also activities like weapon deployment or sensor allocation can be modelled as tasks to be allocated. The use of a flexible workflow system like iTasks gives new possibilities to study and better understand these processes and to implement them.

The applications that support military operations are often dedicated special systems like Combat (Battle) Management Systems. Workflow support with iTasks therefore requires interfacing of iTasks with these systems.

Concluding, the iTasks system is a candidate for use during operations, but the applications are less straightforward than for the planning phase. Pilot studies are needed to investigate the potential of iTasks.

Workflows and Net-Centric Operations

Net-Centric Operations aims to connect parties involved in operations by a communications network that can be used for the exchange of all kinds of information. Not only data and voice information can be exchanged, but also sensor and other operational information. In the ideal situation this leads to shared situational awareness, where all participants in the network have access to relevant information and can deploy appropriate weapon systems in the network. These operations pose great challenges. First of all, connecting all operational partners by a network in an operation is a complex task. Because of the mobility of participants, wireless communication means have to be used. Also the use of different (communication) standards by participants is a problem. But even if these technical problems are solved many challenges remain. Who should get which information at what moment? How is this information represented? What actions are the partners allowed to take? They have to respect (changing) Rules of Engagement. Again a flexible workflow system like iTasks offers a good starting point for the construction of applications for Network-Centric Operations. Other, more mathematical, properties of networks in Net-Centric Operations are discussed in [5].

One of the powers of a web-based toolkit like iTasks is that partners from other countries can participate as soon as they can make a network connection. No special installation of software is needed. This is a powerful feature because partnerships are likely to change on an ad hoc basis.

Future work

The iTasks formalism has a solid theoretical and implementation foundation. Although there is still interesting work to be done on these subjects, the focus for the near future will be on applications. It has already been indicated that iTasks could be useful for the planning phase of complex operations. The working out of a scenario for the planning phase of a realistic operation and the building of a demonstrator application for this scenario is planned. This demonstrator will be used to prove the usefulness of the iTasks approach and to obtain feedback from military experts. It is also expected that the implementation will lead to functionality demands for iTasks.

Current investigations already have lead to new demands for the iTasks system. One of these demands is the possibility of ad hoc creation of new tasks. This means that the user must be capable of creating new tasks (and dependencies between them) while using the workflow system. This on-the-fly task creation will not have the flexibility of the full iTasks system, but should be sufficient for simple sequential, AND, OR and repetitive tasks.

Conclusions

In this paper the iTasks combinator library for the construction of dynamic workflow systems has been described. First, a general description of workflow systems and a justification for applications with a web interface was given. No full description of the iTasks system was given, but instead the system was introduced by a number of simple examples that show the potential of the formalism.

The iTasks system has a number of important advantages in comparison with more traditional workflow formalisms:

- the system has a universal web interface. It can be used without the installation of special software. This allows for the on-the-fly join of new users;
- the system allows for easy security. Security can be based on both shared and public key encryption;
- iTasks has a compact and precise formalism. It allows for the formal reasoning about (dependencies between) activities;
- the formalism is extendable. iTasks has a number of predefined combinators, but it is possible for the application programmer to add new combinators;
- the iTasks formalism is compositional. New combinators can be made by combining existing combinators.

The possible usage of iTasks in the military domain was sketched. The most obvious candidate for its use is the planning phase of complex operations like the deployment of troops for longer periods. This planning involves a large number of people and is

characterised by a highly dynamic nature. Therefore, the traditional workflow tools are less useful. The building of a demonstrator application to investigate the usefulness of iTasks for these planning activities is planned.

It was also indicated that systems built with iTasks can be useful during complex operations and for Network Centric Operations.

References

[1] The Haskell Home Page. www.Haskell.org.

[2] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[3] J.J. Garrett. Ajax. A New Approach to Web Applications. www.adaptivepath.com/ideas/essays/archives/000385.php, visited February 2008.

[4] J.M. Jansen, P.W.M. Koopman and M.J. Plasmeijer. Efficient Interpretation by Transforming Data Types and Patterns to Functions. In H. Nilsson, editor, *Proceedings Seventh Symposium on Trends in Functional Programming*, TFP 2006, Nottingham, UK, volume 7 of *Trends in Functional Programming*. Intellect Publisher, 2006.

[5] R.H.P. Janssen and H. Monsuur. Military Operations Research and Situation Awareness in Networks, NL-ARMS 2008 (this volume), p 73.

[6] M.J. Plasmeijer and P.M. Achten. The Implementation of iData. A Case Study in Generic Programming. In A. Butterfield, C. Grelck, and F. Huch, editors, *Implementation and Application of Functional Languages*, volume 4015 of Lecture Notes in Computer Science, pp 106–123. Springer, 2006.

[7] M.J. Plasmeijer, P.M. Achten and P.W.M. Koopman. iTasks: Executable Specifications of Interactive Work Flow Systems for the Web. In N. Ramsey, editor, Proceedings of the 2007 ACM SIGPLAN International Conference on Functional Programming, Freiburg, Germany, volume ICFP'07, pp 141–152.

[8] Software Technology Research Group, Radboud University Nijmegen. The Clean Home Page. www.cs.ru.nl/~clean.