# Context-aware scientific workflow systems using KEPLER

Anne H.H. Ngu* and Arwa Jamnagarwala

Department of Computer Science,
Texas State University – San Marcos,
601 University Drive, San Marcos, Texas 78666, USA
E-mail: angu@txstate.edu
E-mail: aj1171@txstate.edu
*Corresponding author

George Chin Jr., Chandrika Sivaramakrishnan and Terence Critchlow

Computational Sciences and Mathematics Division,
Pacific Northwest National Laboratory,
902 Battelle Boulevard, Richland, Washington 99352, USA
E-mail: George.Chin@pnl.gov
E-mail: Chandrika.Sivaramakrishnan@pnl.gov
E-mail: Terence.Critchlow@pnl.gov

**Abstract:** Data intensive scientific workflows are often modelled using a dataflow-oriented model. The simplicity of a dataflow model facilitates intuitive workflow design, analysis and optimisation. However, some amount of control flow modelling is often necessary for engineering fault tolerant, robust and adaptive workflows. In scientific domain, myriads of environment information are needed for controlling different stages of execution. Modelling the control flow using inherent dataflow constructs will quickly result in a workflow that is hard to comprehend, reuse and maintain. In this paper, we propose a context-aware architecture for scientific workflows. By incorporating contexts within a dataflow-oriented scientific workflow system, we enable the development of context-aware scientific workflows without the need to use numerous low level control flow actors. This approach results in a workflow that is aware of its environment during execution with minimal user input and that responds intelligently based on such awareness at runtime. A further advantage of our approach is that the defined contexts can be reused and shared across other workflows. We demonstrate our approach with two prototype implementations of context-aware actors in KEPLER.

**Keywords:** context-aware; scientific workflow architecture; context modelling and management in scientific workflow.

**Reference** to this paper should be made as follows: Ngu, A.H.H., Jamnagarwala, A., Chin, G., Jr., Sivaramakrishnan, C. and Critchlow, T. (2010) 'Context-aware scientific workflow systems using KEPLER', *Int. J. Business Process Integration and Management,* Vol. 5, No. 1, pp.18–31.

**Biographical notes:** Anne H.H. Ngu is currently an Associate Professor in the Department of Computer Science at Texas State University – San Marcos. From 1992–2000, she worked as a Senior Lecturer in the School of Computer Science and Engineering, University of New South Wales (UNSW). She has held research scientist positions with Telecordia Technologies and Microelectonics and Computer Technology (MCC). She was a summer faculty scholar at Lawrence Livermore National Laboratory from 2003–2006. Her main research interests are in information integration, service-oriented computing, scientific workflows and agent technologies.

Arwa Jamnagarwala is working as a Research Assistant at the Computer Science Department, Texas State University. She obtained her MS in Computer Science at Texas State University – San Marcos in 2009. Her current research interests include investigating new ways to bring context-awareness within scientific workflow management systems.

George Chin Jr. is a Chief Scientist in the Computational Sciences and Mathematics Division at the Pacific Northwest National Laboratory. He received his PhD and MS in Computer Science from Virginia Tech. His areas of interest include scientific workflows, scientific visualisation, human-computer interaction, social networks and high performance computing.

Chandrika Sivaramakrishnan is a Scientist in the Computational Sciences and Mathematics Division at the Pacific Northwest National Laboratory. She received her MA in Information Systems from Birla Institute of Technology and Science, Pilani, India. Her areas of interest include scientific workflow systems, database management systems, software architecture and content management systems.

Terence Critchlow is the Chief Scientist for Scientific Data Management in the Computational Sciences and Mathematics Division for the Pacific Northwest National Laboratory (PNNL) since 2007. Prior to that, he worked at Lawrence Livermore National Laboratory. He is currently the Thrust Area Lead for Scientific Process Automation area within the DOE SciDAC Scientific Data Management Center, and PI for a DHS S&T data management and analysis project. His current research interests include scientific workflows, data analysis, data integration, metadata and large-scale data management.

## 1 Introduction

Scientific workflow systems aim to provide end-to-end frameworks for automating and simplifying data processing tasks for scientists. These tasks often include data acquisition, transformation, integration, analysis and visualisation. Many existing scientific workflow systems, including KEPLER (Ludäscher et al., 2006), Taverna (Oinn et al., 2004) and SCIRun (Parker et al., 1998) are based on dataflow principles (Lee and Parks, 1995), where individual components (or *actors*) are loosely coupled, communicate via streams of data objects and are scheduled explicitly via the workflow system. Thus, components, which may be native to the system or wrap external components such as web or grid services, scripts or local applications, become reusable components that can be leveraged within various workflows.

Workflows expressed using a dataflow-oriented model can be efficiently analysed and scheduled (Lee and Parks, 1995) and is also a simple and intuitive model for workflow designers (Bowers and Ludäscher, 2005). However, while dataflow has become a popular model, some amount of control flow modelling is often necessary for engineering fault tolerant, robust and adaptive workflows. In particular, myriads of environment information are needed for controlling different stages of execution in scientific workflow. The basic mechanisms currently provided by KEPLER for managing control flow include:

1 allowing actors to have multiple input, output and parameter ports, e.g., long list of ports and parameters

2 allowing 'Boolean-switch' actors to split token streams

3 allowing complex workflow graph structures, e.g., containing cycles, multiple paths, etc.

As discussed briefly in Bowers et al. (2006) and Goderis et al. (2005), modelling control flow using these constructs involves inserting and linking various specialised actors alongside dataflow actors, increasing the complexity of the original workflow and making it difficult to distinguish dataflow from control flow (because they are 'entangled').

In the past, three approaches that allow for flexible modelling and composition of KEPLER scientific workflows such that they enable reuse and repurposing have been proposed. The first approach, frame/template (Bowers et al., 2006), emphasises the decoupling of control flow from dataflow through structured embedding of control flow actors in data-oriented scientific workflows. An abstract actor called *frame* and an abstract workflow called *template* that can be configured at design time were introduced. The main contribution of this particular approach is the encapsulation of complex control flow patterns in a three-layer transducer template that can be modelled via a finite state machine in KEPLER. The frame/template approach allows workflow designers to plug in predefined control flow patterns by simply choosing and applying different frames or templates. This approach introduces some degree of flexibility in the modelling of adaptive workflows at design time, but does not address the adaptive execution of workflow at runtime. For example, once a frame or template has been embedded, different implementations cannot be reselected for embedding at runtime unless the template exhaustively specifies the conditions for embedding all the alternatives and all those alternatives are known a priori.

The second approach is the dynamic frame actor (Ngu et al., 2008). A *dynamic frame* actor is an abstract *higher-order* actor. A higher-order actor is an actor that takes as input the definition of another actor. Dynamic frame extends the frame/template approach by allowing actors to be embedded at runtime, rather than at design time. A dynamic embedding process that allows the automatic generation of an internal model to embed at runtime is proposed. The key innovation in the dynamic frame approach is that we endow the frame actor with a set of rules (embedding logic) that gathers data in the execution environment and selects the relevant concrete actor to embed at runtime based on gathered data. The embedding logic can range from simple conditional statements to complex decisions. The use of dynamic frames eliminates the need for using many low level control flow actors in composing flexible and robust KEPLER scientific workflows. However, because the embedding logic is encoded in the dynamic frame actor, whenever there are conditions in the execution environment that are not encoded in the embedding logic, the dynamic frame actor cannot adapt to them. The embedding logic in the dynamic

frame actor must be re-implemented by a programmer and recompiled to account for new conditions. Moreover, it is not possible to share and reuse the embedding logic in other dynamic frame actors, even in the case where the exact same kind of embedding logic may be applied.

The third approach is called *generic actor* (Scientific Data Management Enabling Technology Center, 2009). A generic actor is an actor that is designed to cover a range of capabilities that can be configured by various configuration files and parameters. The `GenericFileCopier` actor developed at Pacific Northwest National Laboratory (PNNL) is such an example. The `GenericFileCopier` carries a superset of parameters that covers the parameters of all available individual file copier mechanisms in the system. Whenever a user desires to change the destination of an output file that is written using a `GenericFileCopier` actor in a workflow, he/she needs only to change the parameter to adapt the actor to the specific use. However, if the user desires to, for example, save a file to whichever machine on the network that has the lightest CPU load, there is no way to perform this runtime analysis and adapt to it. The other issue with the generic actor approach is that the user can easily be overwhelmed with a long list of parameters where only a few are needed for running the workflow for a specific need.

Generic actor can be viewed as an actor that has all the contextual information embedded in it. The difficulty of developing a one-off context-aware actor as a generic actor is that it lacks a uniform and efficient approach for collecting and applying context data. In this case, the context data and execution logic is embedded in the actor code, making it difficult to evolve an actor's behaviour based on new or changing conditions. Recoding the generic actor every time when one wishes to accommodate more context information or applying the generic actor in a new workflow would be arduous for developers as well as result in a plethora of generic actors. The ability to easily and proficiently identify what context information is collectable and/or available and to modify the behaviour of an actor based on context information would enhance the reusability of any KEPLER actors. Furthermore, the ability to share context information across actors and workflows is important as well because collaborating actors and workflows need a consistent shared view of their computational environment in order to effectively synchronise their actions. Actors or workflows operating on different context views may result in conflicting, discordant or unproductive behaviours as the actors or workflows react to different conditions. The goal of our research and development is to prototype a viable context-aware approach and system capable of addressing these context-centred difficulties and issues.

We hypothesise that to enable actor-oriented scientific workflows to be more personalised, robust and adaptive, data that affects an actor's behaviour should be modelled and managed as context and processed externally by a separate computing unit. For example, in transferring data from one machine to another, a transport actor should automatically choose a specific transport protocol such as bbcp (which stands for point-to-point network file copy protocol) when the size of the file is greater than a certain size and the network speed is slow. The transport actor should not be designed to exhaustively cover all the possible protocols to use for all the potential contexts (user preferences or inputs) that an actor can execute in.

In this paper, *context-awareness* refers to the capability of an actor to be aware of its physical environment or situation and to respond proactively based on such awareness. Technically, a context-aware actor is a type of dynamic frame actor where the embedding logic of the frame is modelled as context and provisioned to the dynamic frame actor as required. The set of context definitions can be changed, updated and incorporated into an actor anytime. Contexts, once defined, can be reused in other dynamic frame actors or other workflows. A context-aware actor can be configured to bind to different contexts and each context can result in the embedding of a different concrete actor at each invocation. The main contributions of this paper are:

- An architecture for context-aware scientific workflow systems such that the context modelling, acquisition and management are decoupled from the scientific tasks.
- The integration of dynamic embedding and a context provisioning system to achieve context-awareness in KEPLER scientific workflows without requiring changes to any of KEPLER's underlying computational models.
- A methodology for the development of context-aware actors that exploit the current execution environment to enable efficient and adaptive execution. We show how this methodology can be applied to the development of two KEPLER context-aware actors.

The paper is organised as follows. Section 2 gives an overview of the KEPLER system and introduces the frame and dynamic embedding concepts that are fundamental to the development of context-aware actors. Section 3 presents the architecture of our context-aware scientific workflow system, the detailed methodology for building context-aware actors and a demonstration of the reusability of existing contexts. Section 4 describes the related work and Section 5 provides some discussion on future work and concluding remarks.

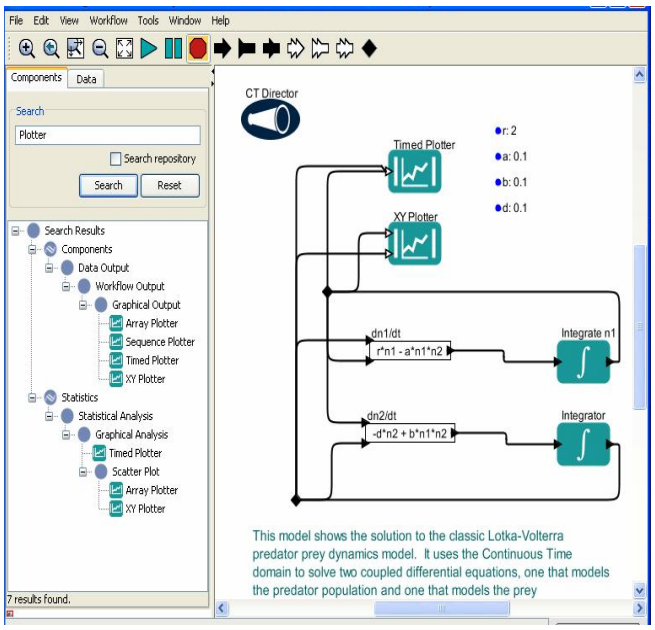
## 2 Background

In this section, we briefly introduce the functionality of the KEPLER scientific workflow system and describe three major concepts that are keys to the development of context-aware scientific workflows.

## *2.1 The KEPLER scientific workflow system*

The KEPLER system provides generic support for designing and executing scientific workflows. KEPLER is built on top of PTOLEMY II (Brooks et al., 2007), which is a system developed for the modelling and design of heterogeneous concurrent systems. In KEPLER, workflows are created by selecting appropriate actors and placing them on a design canvas, after which they can be wired together to form the desired workflow graph. Figure 1 shows the KEPLER user interface loaded with a scientific workflow that implements the classic Lotka-Volterra model of predator-prey population dynamics. The model is represented as a director (labelled 'CT director' in the figure), actors (the boxes) and connections (the arrows) among actors. The actors denote the computational steps and the connections represent data dependencies between steps. The actors labelled `dn1/dt` and `dn2/dt` define two differential equations that are integrated by the two actors decorated with integral symbols when the workflow is executed. The workflow produces graphical renderings of the populations of predators and prey versus time (using the timed plotter actor) and a comparative scatter plot highlighting the interactions of the two populations (using the XY plotter).

**Figure 1** Lotka-Volterra model implemented in KEPLER (see online version for colours)



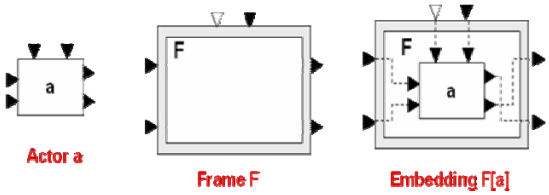

A unique advantage of KEPLER is that the overall execution and component interaction semantics of a workflow is *not* defined by the components, but is factored out into a separate component called a *director* (Eker et al., 2003). A number of dataflow-oriented directors have been implemented, each having certain characteristics. Directors enable different execution models to be used for a workflow and different directors can be used at different hierarchical composition levels.

## *2.2 Actor frame*

Actors in KEPLER are always *concrete* in that they correspond to particular implementations that are directly executed in a workflow. As a simple example, a `gridftp` actor and a `sftp` (secure ftp) actor may be tied to two different data transfer implementations. A *frame* is an abstraction that denotes a set of alternative actor implementations (or refinements) with similar, but not necessarily identical, functionality. For workflow designers, frames are placeholders for components that will be instantiated and specialised at runtime. Thus, a designer can place a frame $F$ on the design canvas and connect it with other workflow components without prematurely specifying which component $C$ is to be used. For component developers, frames can be used as abstractions for a family of components with similar function. For example, we can have a `FileCopier` frame that generalises the transfer of data without specifying whether the implementation is provided by `gridftp` or `sftp`.

Formally, a frame is a named entity $F$ that acts as a placeholder for a component $C$ to be 'plugged into' $F$ (see Figure 2). When devising a frame $F$, a family of components $\mathbf{C}_F$ is envisioned, with each $C \in \mathbf{C}_F$ being a possible alternative for embedding into $F$. Like an actor, a frame has input, output and parameter ports as well as structural types, which together form the *frame signature* $\sum_F$. This signature represents the common API for the family $\mathbf{C}_F$ of components that $F$ abstracts. An *embedding* $F[C]$ of a component $C$ into a frame $F$ is a set of pairs associating (or 'wiring') ports of $C$ with ports of $F$.

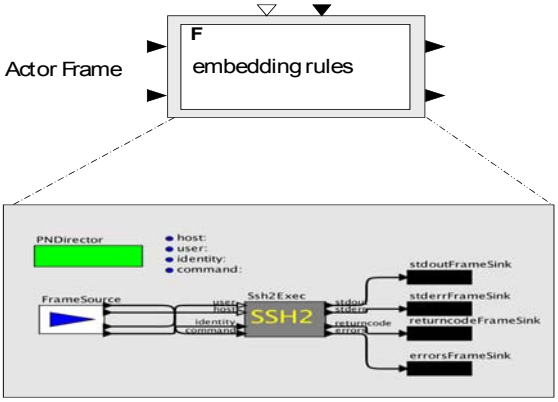
**Figure 2** Embedding of an actor *a* in frame *F* (see online version for colours)




## *2.3 Dynamic embedding*

Frames can be embedded statically or dynamically. In static embedding, a refinement to a frame is embedded during design time. The main problem with static embedding is that if there is a change to a runtime condition, the frame cannot be reconfigured at that time to select a different refinement. In dynamic embedding, the alternative implementations are dynamically selected according to rules employed by the frames themselves at runtime. The refinement to a frame is thus instantiated on demand (Ngu et al., 2008). A type-specific frame actor must implement its own set of embedding rules. Figure 3 shows the generated embedding of an `ssh` (secure shell) at runtime.

**Figure 3** A frame after being embedded dynamically (see online version for colours)




### 2.4 *Context and context-awareness*

Dey and Abowd (1999) define context as 'any information that can be used to characterise the situation of an entity where the entity can be a person, a place or an object that is considered relevant to the interaction between a user and an application including the users and applications themselves'. The two main components in their definition are context and entity. In the context of KEPLER's scientific workflow, the entities are the actors that are the building blocks of the workflows that scientists use. Two types of contextual data are relevant to an actor. The first type is all data that can be *gathered automatically* at runtime to characterise the actor's behaviour. This includes execution location/machine, system parameters (operating system, CPU usage, job queues), system capabilities (availability of certain services or resources), cost of execution (price), temporal information (time of execution) and occurrences of certain events. The second type is data that can be *gathered manually* from a user. This includes his/her preferences such as how results are displayed or delivered; which tools should be used for analysis; privileges and access levels; and preferences for quality of service such as running an actor using the machine with lowest CPU load, transferring data using the most secure protocol or transferring data using the fastest network connection.

Context-aware computing was first presented by Schilit et al. (1994) as 'software that adapts according to its location of use, the collection of nearby objects and the changes of those objects over time'. For example, the Conference Assistant (Dey et al., 1999) is a context-aware application as it uses the conference schedule, attendees' locations and attendees' research interests to suggest presentations for attendees to attend during a conference. In a context-aware application, the burden of gathering and analysing context information is placed upon other computing units. This has the advantage that both the application and the context can be changed independently of each other. For example, if a different context is needed for a context-aware application, the user only needs to define the new context and redo the context assignment with the application. There is no need to change the way the application is implemented. We define context-awareness in KEPLER as the mechanism for adapting the execution of an actor based on the sensed/gathered contextual information. For example, a context-aware `FileCopier` actor in KEPLER can provide flexibility in how a file is being copied based on real-time sensing of the size of the file, the speed of the network connection, and the availability of file transfer protocols on the source and destination machines. The context-aware `FileCopier` can adapt by choosing the 'best' or 'most robust' protocol to use at runtime for the user.

## 3 Enabling context-aware scientific workflow

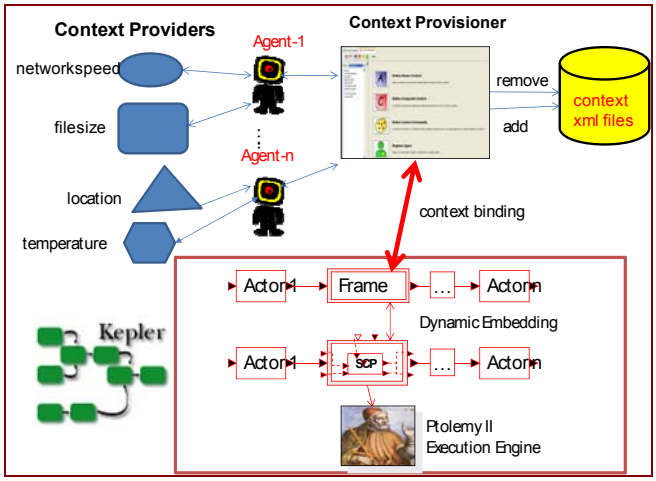
Contextual information in scientific workflow can be gathered from the execution environment (machines), from the users (scientists) or generated by the computation steps (actors) in the workflow. The key challenges in building a context-aware scientific workflow are:

1 the design of a context gathering and analysis engine

2 the design of a context-awareness mechanism that can leverage the gathered contexts at runtime without changing the underlying workflow computation model.

In this section, we describe how we address these two key challenges.

**Figure 4** KEPLER context-aware scientific workflow architecture (see online version for colours)




### 3.1 *Architecture of context-aware scientific workflow system*

The architecture of our context-aware scientific workflow system is depicted in Figure 4. The main components are the Context Provisioner (CP) and the KEPLER scientific workflow system. The CP is a web service that manages

the mechanics of context acquisition and provisioning (i.e., context gathering and analysis engine). It also has an associated graphical editor for easy context definition and management. CP is built on top of the context acquisition engine in the CONTEXTSERV system (Sheng et al., 2009), which is designed for the rapid development of context-aware web services.

### *3.1.1 Context providers*

Context acquisition is performed by various independent context providers. There are different types of context providers. Context providers can range from a hardware device, to a sensor, to a complex computer programme. Each context provider uses a specific technique for context acquisition. For example, context can be acquired by polling, by subscription, by querying a database, by aggregating raw data from sensors or by invoking a web service. The concept of an agent is introduced to provide a uniform abstraction for dealing with the numerous and heterogeneous types of context providers. For example, a web service agent is used for acquiring contexts from all context providers that provide context data via WSDL services. A command line agent is used for acquiring contexts from all context providers that gather data by running various system commands and extract the needed data from the specific output pattern. Agents hide the complexity of context acquisition from developers. Context providers are further divided into local and remote providers. Local providers are those that reside in the local machine. Remote providers are those that require specification of how they can be located and invoked.

### *3.1.2 Context provisioner*

The CP manages context definition and processing of context queries. It contains a set of agents and a repository of context specifications. It is implemented as a web service running on Apache CXF server (a lightweight Java-based web service server). As a web service, it provides five main operations. The operation `getContext` accepts the context query as input and returns the computed value of the context as output. The operation `getContextList` accepts a context query and a condition as inputs and returns a list of contexts that meets the specified query and the condition as output. The operation `saveParameter` is used to pass data from a KEPLER actor to the CP to be saved as named parameters for use in context acquisition. The `reset` operation is used to clear the saved parameters between different context queries. The `getParameter` is used to retrieve the value of a parameter saved by the CP. The context XML files store the specifications of all contexts used by a specific project.

The current system is set up such that each project or type of workflow in KEPLER has its own CP. The CP in every project is the same except for the set of context XML files (i.e., context definitions) stored in its repository and the types of agents that it is associated with.

### *3.1.3 Binding with KEPLER system*

The connection between KEPLER and the CP is established via context binding and context-awareness mechanisms. Context binding is the ability for an actor to bind to or access relevant contexts defined in the CP. For a KEPLER workflow to be context-aware, it must have at least one actor that is implemented as a dynamic frame and must have at least one port or parameter of the frame actor identified as a context-aware object (CAObject). Context-awareness is achieved when the dynamic frame actor is embedded with a refinement in a dynamic embedding process based on evaluation of the associated context.

## *3.2 Building context-aware actors*

The development of context-aware actors consists of three main components depicted in Figure 5. The first component is the modelling and implementation of the frame actor. The second component is the modelling and implementation of the context-awareness mechanism, and the third component is the modelling and implementation of context. In practice, the context modelling should occur first, but for ease of readability of the paper, we will start with the frame actor modelling. We will describe the context-aware actor development process using the `file copying workflow` as a running example. We will first explain the functionality of an existing `GenericFileCopier` actor and its limitations and then discuss how to convert the `GenericFileCopier` actor into a context-aware `FileCopier` actor.

The `GenericFileCopier` actor is designed for scientists to copy files from one machine to another using a variety of file transfer protocols. The way the current `GenericFileCopier` actor works is as follows: the user (scientist) specifies all the configurations of all the machines that the `GenericFileCopier` actor has access to in a machine configuration file. When this actor is instantiated, the user enters the required parameters such as the source and destination machines and the source and the destination files. The destination machine is used to determine which machine's configuration to use. For example, if the user specifies 'zeus.cs.txstate.edu' as the destination machine, then the machine with the specified host name in the configuration file will be matched and the available file transport protocols specified for that host machine (zeus.cs.txstate.edu) will be prompted as a selection list to the user. The main disadvantage of this approach is that it cannot adapt to runtime conditions. For example, it cannot take advantage of runtime information such as size of file or speed of the network to make smart decisions on how to move the data.

**Figure 5** Three main components required for developing context-aware actors (see online version for colours)

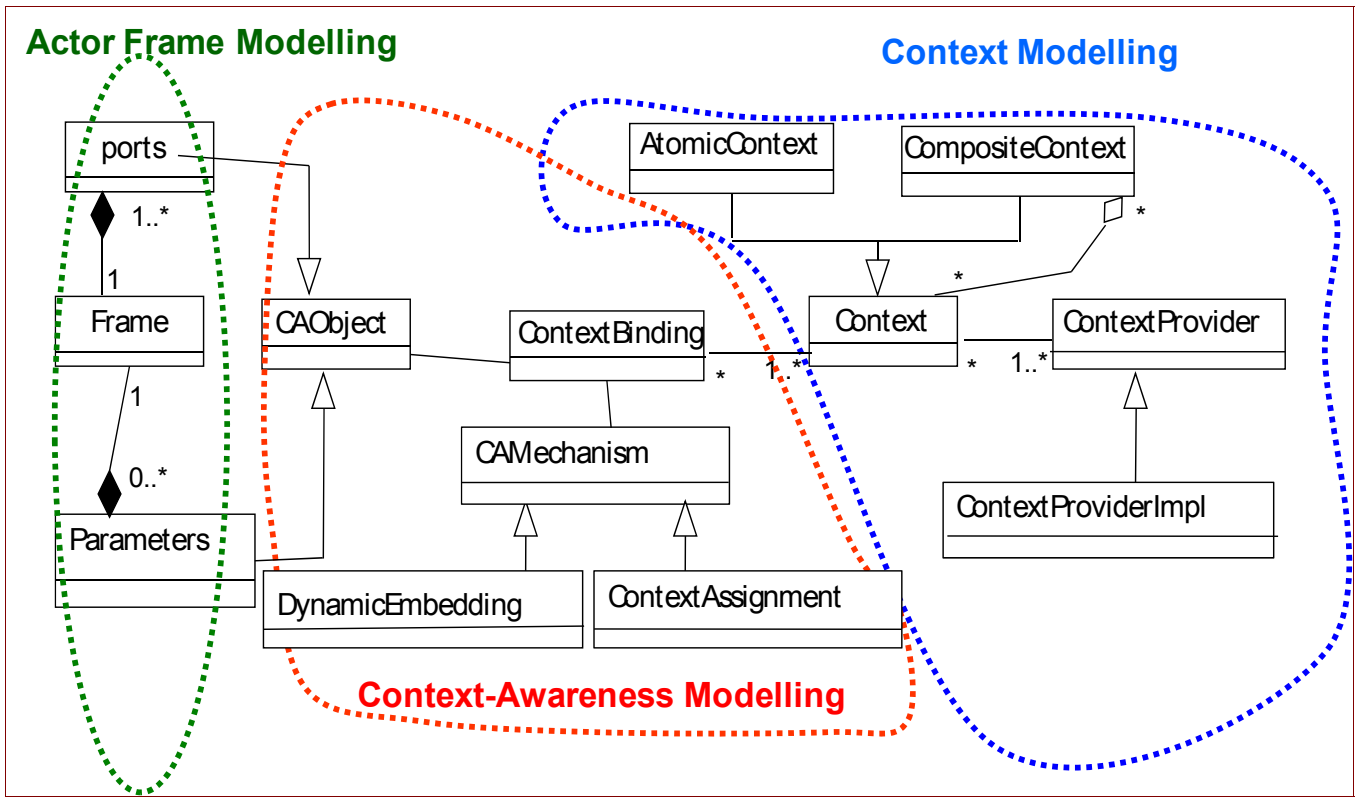
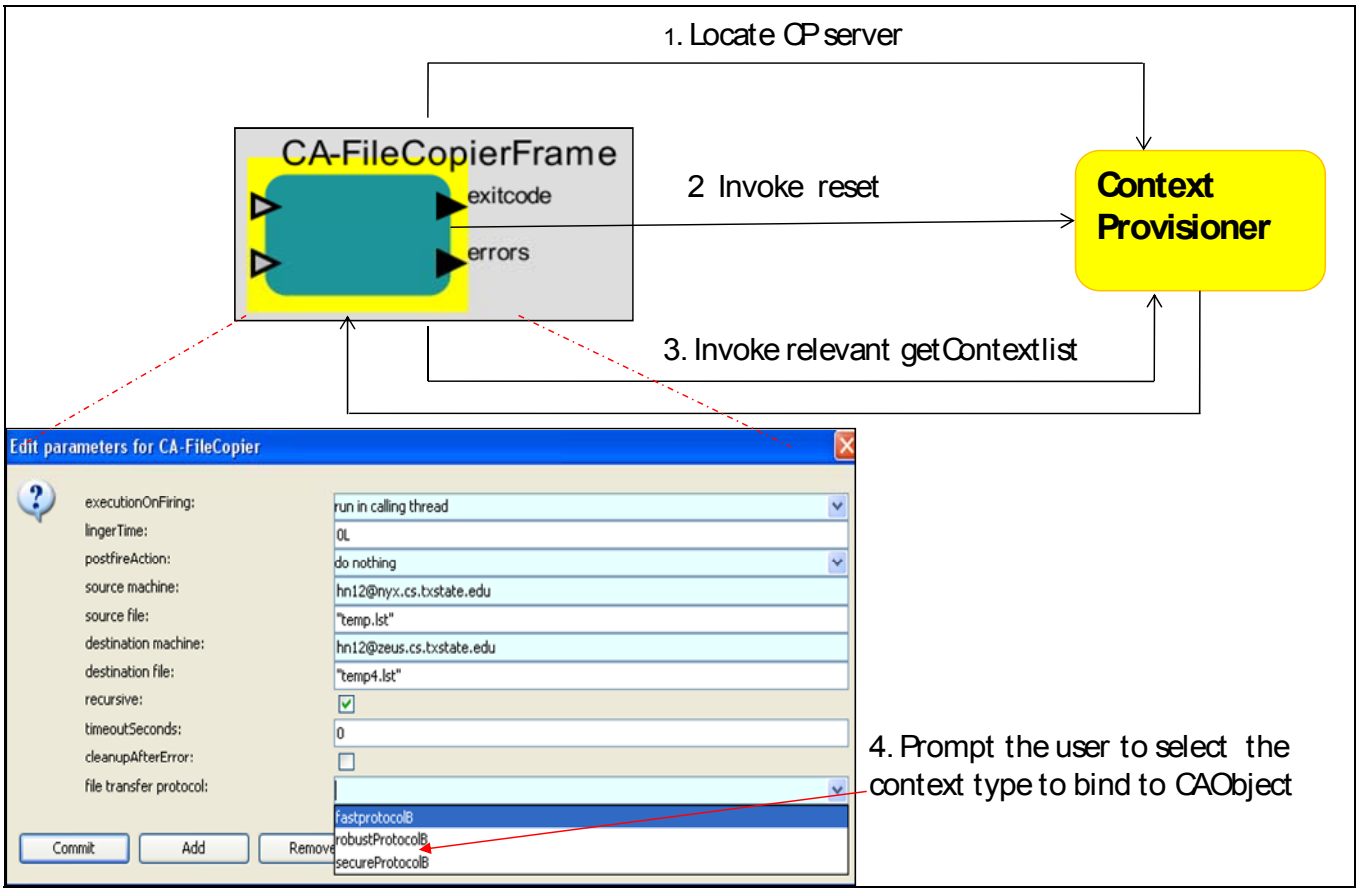



**Figure 6** Context binding process (see online version for colours)



*Actor frame modelling* component consists of modifying the `GenericFileCopier` actor to be a frame by making it a subclass of the frame actor. Because a frame is also an actor, the parameters and ports that were defined for the generic actor can be reused. The only additional step required is the identification of which port or parameter is context sensitive and to tag it as a CAObject. Some low level parameters such as those used to obtain file transfer protocol instalation path and command option can be inferred by the CP during context acquisition process and thus deleted from the context-aware `FileCopier` actor as shown in Figure 6.

The *context-awareness modelling* step consists of the design and implementation of both context binding and context-awareness mechanisms. Context binding involves:

1 locating the CP server

2 invoking a `reset` operation

3 sending a query to the CP to get the list of relevant contexts for the CAObject

4 prompting the user to choose a specific context to bind with the given CAObject.

The frame actor developer must implement the semantics of the relevant contexts for a particular CAObject. For example, in the context of the `file transfer` protocol CAObject, the relevant contexts are all the contexts that can be used for making the recommendation of a specific transfer protocol to use at runtime. Figure 6 shows an example of a context-aware `FileCopier` actor binding to a context called *fastProtocol*.

**Table 1** Implementation of context-aware `selectActor()`

```
ComponentEntity selectActor()
   ComponentEntity container = null;
   process dynamic frame actor's input
   (para1 … paran)
   invoke("saveParameter", para1 … paran);
   process dynamic frame actor's CAObject;
   selectedActor = invoke ("getContext",
   CAObject);
   container = selectedActor;
   ..
```

The key to the successful integration of frame and context is in the encapsulation of embedding logic as context that can be defined and managed separately from the frame actor. A dynamic frame actor must implement a method called `selectActor()`, which is responsible for generating a model to execute the selected concrete actor at runtime. The implementation of context-awareness involves adding support to the `selectActor()` method to query the CP to process the requested context information, which in this case is fastProtocol. The response from the CP is used to determine how the frame actor should be adapted at runtime. Table 1 shows the pseudo code of the `selectActor()` method for the context-aware `FileCopier` actor. This will result in the recommendation of a preferred protocol taking into account the contextual information such as size of input file, availability of protocols and speed of network connection.

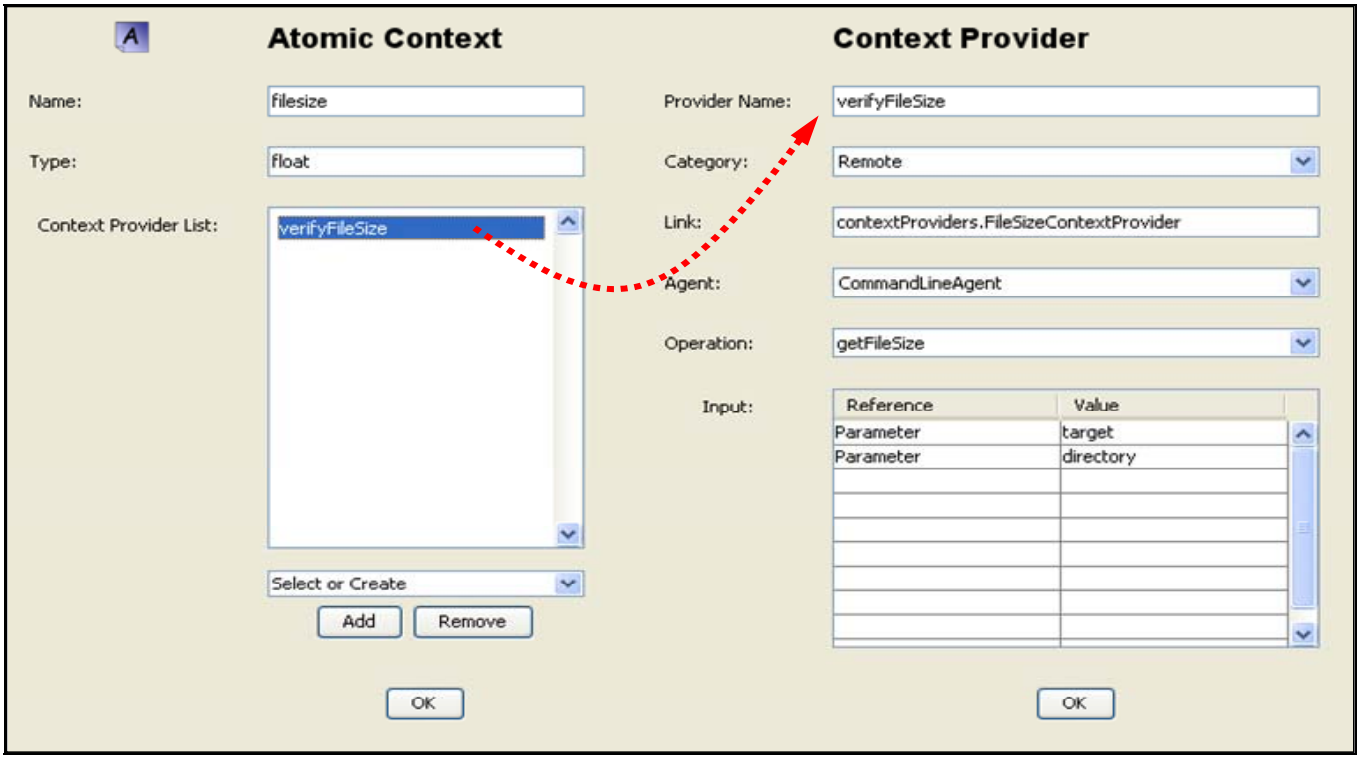
The *context modelling* step deals with the definition of contexts and the associated providers. The main classes used for context modelling are shown on the right side of Figure 5. Context can be modelled as either an atomic or a composite context.

Atomic context is the primary context that does not rely on other contexts and is acquired directly from a context provider. We identified the following three atomic contexts for the context-aware `FileCopier` actor: `filesize`, `networkspeed` and `protocol availability`. These are data that will affect the behaviour of the `FileCopier` actor. They can be gathered automatically once some parameters, such as the source and destination machines, are known. Figure 7 shows how the `filesize` atomic context is defined through the CP's graphical editor. The main elements of atomic context are `name`, `type`, `category`, `agent` and `context_provider`. Table 2 shows how it is represented and stored in an XML context file. The `name` serves as an identifier for this specific context type. The `type` float represents the data type of this context type. The `context_provider` describes how the context is acquired and via what operation, what parameters and what types of agents. The right side of Figure 7 shows how a context provider called `verifyFileSize` is specified. It is the responsibility of the context provider to provide the implementation of how to check the size of any given file in any given machine.

**Table 2** XML representation of atomic context

```
<context>
   <name>filesize</name>
   <type>float</type>
   <category>Atomic</category>
   <context provider>
      <name>verifyFileSize</name>
      <category> Remote</category>
      <agent>commandLineAgent</agent>
      <link>contextProviders.FileSizeContextprovider</link>
      <operation>getFileSize</operation>
      <input ref=<Parameter> target</input>
      <input ref=<Parameter>directory</input>
   </context provider>
</context>
```

**Figure 7** Atomic context specification (see online version for colours)

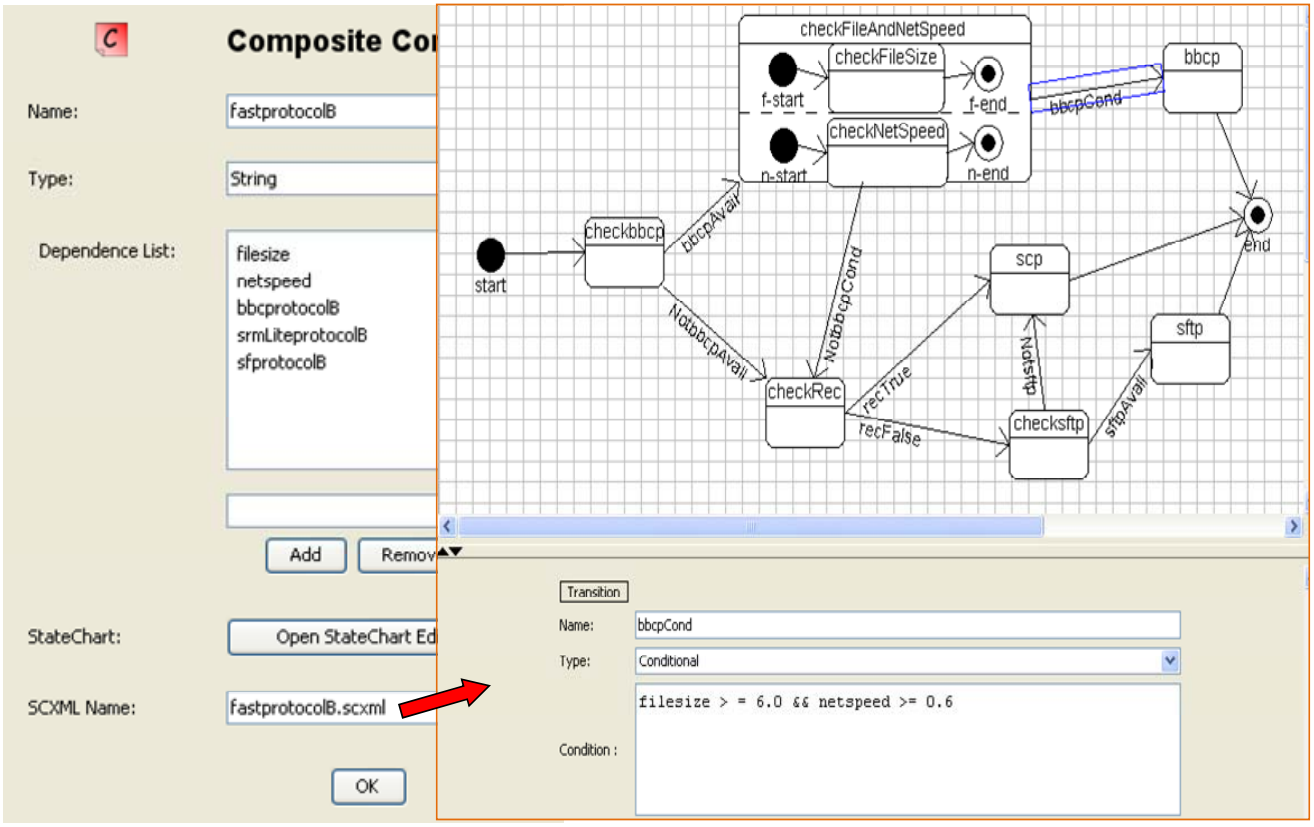



Composite context is an aggregation of multiple atomic or composite contexts and cannot be acquired directly from a context provider. Composite context combines multiple contextual information and arrives at a decision based on these aggregated contexts. The three main composite contexts defined for the context-aware `FileCopier` actor are `fastProtocol`, `robustProtocol` and `secureProtocol`. Each of the above contexts represents the implicit rules that a typical user would apply to choose a particular protocol to use when moving data from one machine to another. Table 3 shows a summary of the `fastProtocol` rule expressed in an if-else rule format. The main elements of a composite context are: `name`, `type`, `category`, `dependent contexts` and `SCXML file name`. The details of how the dependent contexts are aggregated are described using a state chart. SCXML is a W3C-supported XML-based standard for describing state charts. Figure 8 shows how the `fastProtocol` if-else rule is defined in an UML state chart. The `fastProtocol` state chart has an initial state, a final state, a set of simple states, one composite state that has two parallel substates nested inside it and a set of transitions. After the state chart is modelled, it can be exported as an SCXML file. This SCXML file can be executed by an open source SCXML engine to obtain the aggregated context value.

**Table 3** If-else rules for deciding fast file transfer protocol

```
if (bbcp protocol is available) {
    if ((filesize is >=6 GB) and (network speed > =6ms))
            choose bbcp
    else if ((filesize < 6 GB) or (networkspeed < 6 ms))
        if (recursive transfer is true)
            choose scp protocol
        else
            if (sftp protocol is available) choose sftp protocol
            else default to scp protocol
else (bbcp protocol not available)
        if (recursive transfer is true)
            choose scp protocol
        else
            if (sftp protocol is available) choose sftp protocol
            else default to scp protocol
}
```

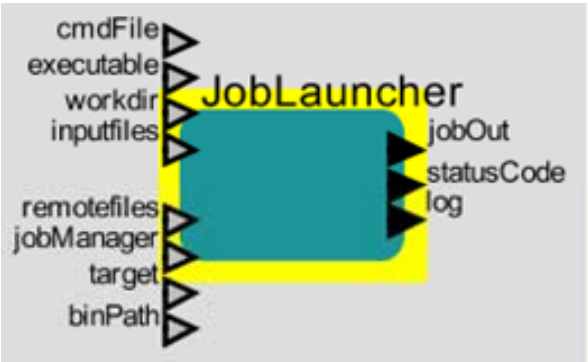
**Figure 8** Composite context specification (see online version for colours)




## 3.3 *Reusability of contexts*

One of the key advantages of our context-aware scientific workflow architecture is the reusability of defined contexts and the ability to automatically gather contextual data, which reduces the complexity of actor design. We demonstrate in this section how the context-aware `RemoteExecution` actor can reuse contexts defined in the context-aware `FileCopier` actor. We also show that the context-aware `RemoteExecution` actor can be designed with fewer parameters while achieving the same functionality as a traditional `RemoteExecution` actor.

**Figure 9** KEPLER `jobLauncher` actor (see online version for colours)




The goal of a `RemoteExecution` actor is to allow a user to submit a job to be run on a remote high performance computer. The main inputs to this actor are the name of the remote machine, the job manager to use and the script for submitting the job. A traditional `RemoteExecution` actor assumes that the user has prior knowledge regarding the correct job script to use for the chosen job manager and also the presence of a particular job manager on the chosen remote machine. Setting up a `RemoteExecution` actor that can check the combination of all different correct inputs will lead to a very complex actor that is hard to reuse. Figure 9 shows all the parameters that need to be configured to run a `jobLauncher` actor, which is a generic version of a `RemoteExecution` actor. Moreover, this `jobLauncher` actor does not address higher-level user choices such as submitting a job to a remote machine with the lightest CPU load or shortest job queue. A context-aware `RemoteExecution` actor enables the user to submit the job based on higher-level concepts such as submitting a job to the machine with maximum idle time, maximum connection speed or minimum job queue. It also eliminates the need for the user to specifically define which job manager, binPath and executable to use. All of these parameters can be gathered as context given the identity of the target machine.

Following the methodology for building context-aware actors described in Section 3.2, the actor frame modelling resulted in identifying the user preference for the remote machine as a CAObject.

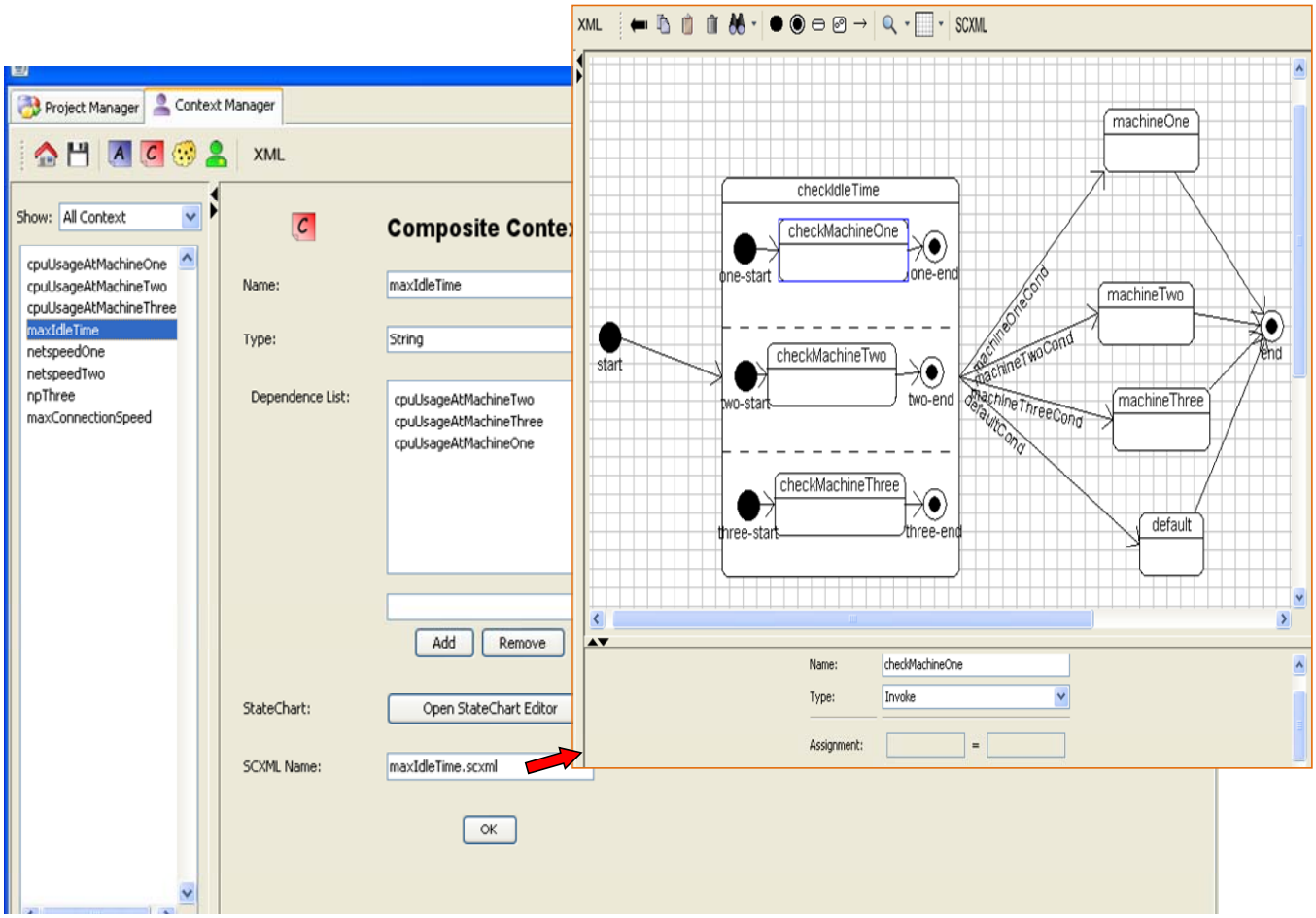
The context-awareness step consists of context binding and dynamic embedding. In context binding, the list of potential remote machines that the user can access is passed to the CP using the `saveParameter` operation and the user preference CAObject is bound to a list of relevant contexts by looking up all contexts that are related to machine performance. In our example, the relevant contexts correspond to `maxIdleTime`, `maxConnectionSpeed` and `minJobQueue`.

In the context modelling step, the two new atomic contexts `cpuUsage` and `jobQueue` are defined. `cpuUsage` will return the idle CPU percentage from a `vmstat` command of a particular Linux remote machine. `jobQueue` returns the percentage of active jobs over the total number of jobs in the queue. Active jobs are those that are running, on hold or pending. The atomic context `networkspeed` is reused from before.

Three composite contexts that reflect three different user preferences for a remote machine are defined. The `maxIdletime` context that determines the machine with the least CPU usage is modelled using the state chart as shown in Figure 10. In this state chart, it checks three given remote machines in parallel. The transition specifies conditions for selecting the machine with the highest idle time. Similar logic is used to select the machine with the highest network connection speed and the lowest job queue. Here, we demonstrate the reusability of the `networkspeed` atomic context and the composite context composition logic.

Unlike the traditional `RemoteExecution` actor, the context-aware version enables the actor to adapt to the latest machine conditions. The user is not required to have prior knowledge on which the job manager is running on which remote machine or the location of job manager bin path, etc., to submit a job.

**Figure 10** Composite context for determining the machine with highest idle CPU (see online version for colours)




## 4 Related work

In mobile and pervasive computing domains, context-awareness is seen as one of the key factors for the successful deployment of many of their applications. Some of the best-known context-aware mobile applications are location-based services and various intelligent assistant services. Two recent surveys on context-aware systems can be found in Baldauf et al. (2007) and Kapitsaki et al. (2009). The survey by Kapitsaki et al. (2009) covers the different approaches for handling context in an application; these

approaches range from source code level programming, model driven approach, to message interception. The second survey (Baldauf et al., 2007) focuses on the design principles of context-aware systems. The authors advocate a layered architecture with context acquisition, context-awareness and context modelling as distinct layers.

Our work is inspired by the phenomenal growth in mobile services and applications especially after the successful launch of the iPhone by Apple Computer. In mobile applications, the environment that users interact with is dynamic by nature (changing location, changing bandwidth, changing battery power, changing time zone). It is impossible to enumerate all the possible changes a priori in mobile applications. This calls for an agile software design methodology where context modelling, managing and provisioning are central to the development of mobile applications, yet are decoupled from the business logic of mobile applications. Many context-aware applications have recently been successfully deployed (e.g., GPS, ActiveBadge and Forerunner). Our work is further influenced by the success of CONTEXTSERV (Sheng et al., 2009), which is a platform for the rapid development of context-aware web services using a model driven approach called *contextUML* (Sheng and Benatallah, 2005). We compare our context-aware scientific workflow framework to some recent context-aware or adaptive systems in terms of context modelling and management, awareness mechanism and extensibility of the architecture.

The awareness-enabled coordination (AEC) (Georgakopoulos et al., 2006) is a process collaboration and automation platform designed to scale to large multi-organisation teams. It was an extension of CMI (Rusinkiewicz and Georgakopoulos, 1999), an advanced process automation research prototype. AEC has demonstrated the benefit of context modelling and a contextualisation mechanism for supporting unrestricted dynamic changes in processes without the need to model those changes using explicit control flows. In AEC, context is a mosaic of information, knowledge, resources, teams, organisations, goals and events that are gathered for a particular collaboration situation. Each context in AEC consists of a *scope* and a set of context elements. The context elements include policies, resources and methods with meaning local to the process or workflow. Context within AEC can form a context network via referential relationships. AEC context requires the implementation of a full ontological model. Our context is based on a simple XML markup scheme. We leverage UML state charts to relate and aggregate multiple contexts. The complexity needed to set up and model contexts using a full ontological model incurs significant overhead in scientific workflows that typically do not require coordination across different teams or organisations. The AEC awareness mechanism is based on a complex event processing system with extensibility provided via new event operators.

Context-aware adaptable web services (Keidl and Kemper, 2004) are one of the early representative works proposed for the development of innovative web applications that can response proactively and intelligently based on users' context. In this framework, contexts are limited to the information of service requesters and are embedded in simple object access protocol (SOAP) messages. Context-awareness is achieved by intercepting SOAP messages and invoking the matching context plug-ins or context services. Context acquisition and management are not supported in this system. Thus, context specification cannot be shared and reused in other web services. Moreover, their context-awareness mechanism is limited to pre- and post-processing of input and output of SOAP messages. Our CP allows a developer to define a variety of contexts (user preferences, environment parameters, resources, policies) and allows context to be shared, aggregated and exchanged in a distributed fashion. We also provide a graphical editor to ease the definition and maintenance of contexts.

In Bowers et al. (2000), an adaptive approach to developing survivable and robust applications is described. The approach allows systems to adapt to changing environments so that they may continue to perform even with reduced functionality in cases of resource constraints or compromised conditions. They introduce adaptation spaces as a way to model the dynamic and changing environment contexts. An adaptation space is a specification of the condition space (can be a cross-product or a subset of the conditions of interest) that will be evaluated in order to select the appropriate implementation strategy in a dynamic execution environment. The adaptation space allows arbitrary policies or conditions to be specified using a simple XML markup scheme. The adaptation space also allows the specification of preferences of the implementation strategy from the user's perspective. An *adapt* operator is used to parse the adaptation space specification, evaluate the conditions and select the alternative implementation given a set of user preferences. Adaptation space specification has similar representation power as our composite context. An adaptation space must be defined for each individual application and it is unclear how different adaptation spaces can be composed, reused and shared across different applications. Only a single awareness mechanism is provided via the *adapt* operator. Within the adaptation space architecture, context acquisition and management are not discussed.

In the CoreGRID project (Caeiro-Rodriguez et al., 2008), dynamicity is viewed as one of the most important requirements for scientific workflow management systems. The authors describe the inherent need to support changes during execution of workflows because not only do scientific exploration processes evolve, but so do networks, platforms and resources that support execution of the processes. The proposed solution is a dynamic mapping of workflow specification to changing availability of resources. A workflow specification is created with placeholder and semantic tasks where binding of tasks to resources is accomplished at runtime. This mechanism for achieving adaptive workflows is similar to our earlier approach of frame and dynamic embedding (Ngu et al.,

2008). Context-aware scientific workflows can achieve the same level of dynamicity with the added advantage of being able to model the changing environment as contexts that can be reused across different workflows.

Wieland et al. (2007) propose the concept of context-aware workflows and present an implementation of this concept using an extended workflow language called Context4BPEL. The Context4BPEL execution environment consists of the BPEL workflow engine and the context event scheduling system working in collaboration with the Nexus context management platform. The Nexus platform serves as the CP and provides support for context queries, context events and context services. A Smart Factory application is prototyped with this framework. The context data, such as availability of tools, machines, workers and stock amount, are collected and saved via tags in the Smart Factory and are used to drive the real-time execution of workflow by observing the state of the factory. Context4BPEL is tightly coupled with the context provisioning system. It is not clear how a context-aware BPEL workflow can be bound with different contexts. Our context-aware workflows allow the same workflow to be bound with different contexts depending on the users' needs.

Aspect-oriented programming (AOP) (Murphy and Schwanninger, 2006) is emerging as a promising technique for realising adaptive and flexible web services composition. Non-functional composition properties (e.g., service selection policies and business rules) can be specified as aspects, which can be activated or deactivated appropriately at execution time. However, applying AOP to web services composition is still in its early stage. The hooks for plugging the appropriate aspects at runtime are limited to what is expressible in XPath. It is still very tedious and complex for users to specify aspects using XML or Java classes. Our context modelling provides a much higher-level abstraction for achieving the similar goal.

## 5 Conclusions and future work

A context-aware application has the capability of being aware of its physical environment or situation (context) and responding proactively and intelligently based on such awareness. Context-awareness is one of the most important trends in computing and is becoming more important with the relentless growth in mobile devices and services. However, in the scientific workflow domain, context-awareness has not been exploited. Contextual information is usually encoded in the process as various control flows, which make the workflow hard to reuse and comprehend. We have proposed a high level abstraction called *frame* and an adaptive mechanism called *dynamic embedding* in the past to enable flexible modelling of scientific workflows. In this paper, we proposed the integration of a context provisioning system with a dynamic embedding process to achieve context-awareness in KEPLER scientific workflows. We proposed a context-aware architecture for scientific workflows and three basic components required for the development of context-aware actors. We illustrated the context-aware actor development process using the `FileCopier` actor and the `RemoteExecution` actor. We also demonstrated that the same context definition can be reused in different context-aware actors.

In our context-aware architecture, context, context providers and frame actors are defined and managed autonomously from each other. This means they can be changed independently. New contexts can be added anytime by setting up new definitions and associating them with context providers. Context providers can change the techniques they use to acquire context information anytime without affecting the context definitions. The frame actor can change the context-awareness mechanism without affecting the context provisioning process.

When compared to a generic actor, a context-aware actor minimises the number of parameters or ports that are needed for the user to interact with. Through some existing inputs, other necessary input data can be gathered at runtime. For example, with `remoteExecution` actor, the system can automatically figure out which job manager is running on the chosen remote machine and provide the latest status on the job queue, rather than asking the user to specifically enter the type of job manager to use for a particular job submission instance. Moreover, contexts hide the complexity of using the workflows by replacing a number of low level parameters with one single high level concept.

When compared with the dynamic frame actor, a context-aware actor can bind to context that determines precisely the set of embedding rules that can be applied at runtime to adapt the actor to changing runtime conditions. When this actor is run in a different environment or by a different scientist, different embedding logic can be applied by choosing a different context binding.

The current implementation of the CP is a synchronous web service invoked by a dynamic frame actor at runtime. We intend to improve the current design in two significant ways in the future. First, we want to provide asynchronous communication between the KEPLER system and the CP. This would enable context-aware actors to perform useful operations in parallel while waiting for data from the CP. For example, a KEPLER actor might want to be informed when contextual information it depends on has been modified.

Second, we want to support other context binding and awareness mechanisms. Currently, context binding and awareness is implemented only using a dynamic frame and dynamic embedding process. The dynamic frame design works best when context has to be shared between actors from the same family group. For example, a generic file copier actor can be created as a superset of a scp copier actor and a sftp copier actor. The generic file copier can then be implemented as a dynamic frame that interacts with the CP and binds with the appropriate file copier at runtime.

This approach, however, might not be best for all scenarios. For example, consider the scenario in which log-in credentials have to be shared between a file copier actor and a job submission actor. Currently, to make these

two actors aware of the common credential information, we need to change both actors to be dynamic frame actors. Each actor would have the same core logic for context gathering and differ only in the way the frame's output port is mapped to the actual actor's input port. Such use cases can be handled better by introducing a different context binding and awareness mechanism that does not require an actor to be a dynamic actor. This will indirectly enable our CP to be used with other scientific workflow systems. We will investigate how the existing context provisioning infrastructure can be combined in more than one way to capture and reuse different types of context information needed by the diverse types of scientific workflows.

## Acknowledgements

We would like to thank our colleagues in the KEPLER Open Source Community for their support. We would also like to thank Dr. Sheng and Ms. Hoi Sim Wong from the Adelaide University, Australia, for making the source of CONTEXTSERV available for us to use. The funding for this work was provided by the DOE SciDAC Scientific Data Management Center.

## References

Baldauf, M., Dustdar, S. and Rosenberg, F. (2007) 'A survey on context-aware systems', *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4, pp.263–277.

Bowers, S. and Ludäscher, B. (2005) 'Actor-oriented design of scientific workflows', in *ER*, pp.369–384.

Bowers, S., et al. (2000) 'Applying adaptation spaces to support quality of service and survivability', in *DARPA Information Survivability Conference and Exposition (DISCEX 00)*.

Bowers, S., Ludäscher, B., Ngu, A.H.H. and Critchlow, T. (2006) 'Enabling scientific workflow reuse through structured composition of dataflow and control flow', in *SciFlow*.

Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y. and Zheng, H. (2007) 'Heterogeneous concurrent modeling and design in Java, volume 1: introduction to Ptolemy ii', Technical Report Technical Memorandum UCB/EECS-2007-7, Univ. of California, Berkeley.

Caeiro-Rodriguez, M., Priol, T. and Nemeth, Z. (2008) 'Dynamicity in scientific workflows', Technical Report CoreGRID Technical Report, TR-0162, Institute on Grid Information, Resource and Workflow Monitoring Services (CoreGRID), August.

Dey, A. and Abowd, G. (1999) 'Towards a better understanding of context and context-awareness', Technical Report GIT-GVU-99-22, iGVU Center, Georgia Institute of Technology, June.

Dey, A., Salber, D., Abowd, G. and Futakawa, M. (1999) 'The Conference Assistant: combining context-awareness with wearable computing', in *Proceedings of the 3rd IEEE International Symposium on Wearable Computers (ISWC99)*, Washington DC, USA.

Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S. and Xiong, Y. (2003) 'Taming heterogeneity – the Ptolemy approach', *Proc. of the IEEE*, Vol. 91, No. 1.

Georgakopoulos, D., Nodine, M., Baker, D. and Cichocki, A. (2006) 'Awareness-based collaboration driving process-based coordination', in *Proceedings of the Second International Conference on Collaborative Computing (CollaborateCom 2006)*, Atlanta, Georgia.

Goderis, A., Goble, C., Sattler, U. and Lord, P. (2005) 'Seven bottlenecks to workflow reuse and repurposing', in *ISWC*.

Kapitsaki, G.M., Prezerakos, G.N., Tselikas, N.D. and Venieris, L.S. (2009) 'Context-aware service engineering: a survey', to appear in *The Journal of Systems and Software*.

Keidl, M. and Kemper, A. (2004) 'Towards context-aware adaptable web services', in *Proceedings of the Thirteen International Conference on World Wide Web, WWW2004*, New York, New York.

Lee, E.A. and Parks, T.M. (1995) 'Dataflow process networks', *Proc. of the IEEE*, Vol. 83, No. 5, pp.773–801.

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Jones, M., Tao, J. and Zhao, Y. (2006) 'Scientific workflow management and the Kepler system', *Concurrency and Computation: Practice & Experience*.

Murphy, G. and Schwanninger, C. (2006) 'Aspect-oriented programming', *IEEE Software*, Vol. 23, No. 1, pp.20–23.

Ngu, A., Bowers, S., Haasch, N., McPhilips, T. and Critchlow, T. (2008) 'Flexible scientific workflow modeling using frames, templates and dynamic embedding', in *Proc. of the Intl. Conf. on Scientific and Statistical Database Management (SSDBM08)*, July.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A. and Li, P. (2004) 'Taverna: a tool for the composition and enactment of bioinformatics workflows', *Bioinformatics*, Vol. 20, No. 17.

Parker, S.G., Miller, M., Hansen, C.D. and Johnson, C.R. (1998) 'An integrated problem solving environment: the SCIRun computational steering system', in *HICSS*.

Rusinkiewicz, M. and Georgakopoulos, D. (1999) 'From coordination of workflow and group activities to composition and management of virtual enterprises', in *Proc. of the Intl. Symposium on Database Applications in Non-Traditional Environments*.

Schilit, B.N., Adams, N. and Want, R. (1994) 'Context-aware computing applications', in *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp.85–90, IEEE Computer Society.

Scientific Data Management Enabling Technology Center (2009) *SDM Center FY 2009 Annual Report*, October.

Sheng, Q. and Benatallah, B. (2005) 'ContextUML: a UML-based modeling language for model-driven development of context-aware web services', in the *Proceedings of the 4th International Conference on Mobile Business (ICMB05)*, Sydney, Australia.

Sheng, Q., Pohlenz, S., Yu, J., Wong, H., Ngu, A. and Maamar, Z. (2009) 'ContextServ: a platform for rapid and flexible development of context-aware web services', in *Proceedings of the 31st International Conference on Software Engineering (ICSE 2009)*, Vancouver, Canada.

Wieland, M., et al. (2007) 'Towards context-aware workflows', *CAiSE'07 Proceedings of the Workshops and Doctoral Consortium Vol. 2*, Trondheim, Norway, 11–15 June.