Infrastructural support for data dependencies in data-centered software systems

Lieven Desmet, Frank Piessens, Wouter Joosen
DistriNet, Dept. of Computer Science, Katholieke Universiteit Leuven

Lieven.Desmet@cs.kuleuven.ac.be

1 Introduction

The identification of key concerns is crucial for a good application of the separation-of-concerns principle [5, 3]. However, an exhaustive list of all important non-functional concerns and the correct decomposition of software into those concerns is still an open question. Moreover, we believe that some of the important key concerns are application domain or software architecture specific.

Therefore, we argue that in order to provide better infrastructural support, the infrastructure must take into account this architectural correlation. The infrastructure must provide explicit support for describing and enforcing implicit application information, that is specific to the software architecture.

In this paper, we illustrate this idea for the concern of data dependencies in data-centered software systems.

2 Data flow dependencies

In the data-centered architectural style [6], a system consists of a central data structure (representing the state of the system) and a set of separate components interacting with the central data store. The components of a data-centered software system describe a *required* and *provided* dataset, specifying the set of data that a component fetches from or puts onto the shared repository. A correct composed data-centered application is a collection of separate components and a shared repository, with respect to functional data dependencies: every required data item of a component is provided by another component by means of the shared repository.

Figure 1 illustrates a simple, servlet-based [4, 2] e-commerce web application. Three different services are identified within the application: adding a product item to the personal shopping basket, the payment of the shopping order and searching through the website. Each box represents a functional task implemented as a servlet, and the services are pipe-and-filter compositions of several independent tasks. The functional data dependencies are depicted by dotted lines. For each website user, for example, a personal shopping basket is saved at server-side, in the session

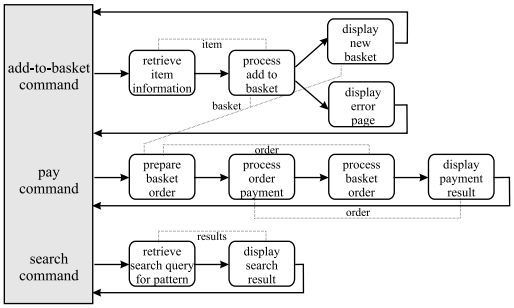


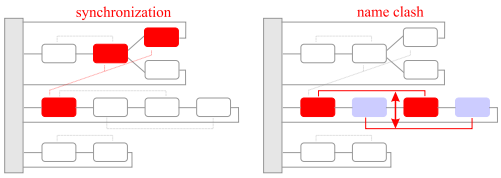
Figure 1: A small e-commerce web application

scope of the shared repository. The personal basket is created at a user's first visit and it is used by three different servlets, entitled in the figure as 'process add to basket', 'display new basket' and 'prepare basket order'.

Besides functional data dependencies, also non-functional requirements on the dataflows may exist. In other words extra constraints on the dataflows through the shared repository may be expressed. These constraints can address for instance the authenticity of the dataflow, confidentiality or synchronization. For instance, in order to prevent race conditions, the composed application needs also extra synchronization support for the shopping basket (figure 2(a)).

Also more general constraints, such as splitting up the shared repository in several disjunct logical repositories, or protecting the repository against name clashes between several dataflows are possible extra composition requirements in data-centered applications. For example, within the payment service, two dataflow dependencies are present. Since the servlets of this service are developed separately (e.g. the payment component is typically a third-party component), a conflict exists in the naming of the shared data. Therefore, extra support is needed to prevent the name clash (figure 2(b)).

From our point of view, data dependencies should be explicitly modelled by the software composer in data-centered application to enforce correctness. Furthermore, data dependencies should be clearly separated from the core functionality in order to improve reusability, adaptability and



(a) synchronization (b) name clash

Figure 2: Data dependency constraints

manageability. In the following section, appropriate infrastructural support for data dependencies is outlined.

3 Infrastructural support

We believe that an infrastructure must provide explicit support for describing and enforcing implicit constraints and dependencies. Therefore, in order to introduce specific support for implicit data dependencies, we have identified the following approach in which three requirements can be defined. Firstly, functional components within the application need an explicit specification extension of the required and provided data items for each component. Secondly, composing an application requires a declarative policy of the functional and non-functional data dependencies between the components. Finally, the enforcement of this declarative policy is needed, either at deployment time or at run-time.

Extended specification Traditionally, an operation is syntactically and semantically specified based on the operation's name and its input and output. In order to express data dependencies, this specification must be extended with extra information about the data items that are provided or required from the shared repository by the component's operation. Extending the specification with repository interactions can be either done manually by the component's designer or implementor, or can be generated by tools based on the component's implementation.

Declarative policy The composition of an application with a shared repository requires more than defining the functional components within the application and the corresponding control flow. The application composer also needs to define the dataflow by means of functional data dependencies and non-functional constraints on the dependencies. Moreover, to enhance adaptivity and manageability, the dataflow should be described in a separate, declarative policy.

Policy enforcement To enforce the dataflow policy at the infrastructure, additional support is needed for controlling the access to the shared repository. As such, the shared repository can be extended with an enforcement engine. Alternatively, a wrapper with a built-in enforcement engine can encapsulate all access to the shared repository. The enforcement engine decides whether a component is allowed or denied access to a data item on the shared repository (functional data dependency). In addition, the enforcement engine ensures the non-functional constraints on the considered dataflow.

In the presented approach, both functional and non-functional data dependencies are clearly separated from core functionality and existing specification of the different components. Furthermore, the data dependency concern is easily adaptable through the use of the declarative policy, and together with the enforcement engine a high cohesion and low coupling is achieved.

4 Current status

Currently, the approach for describing and enforcing data dependencies has been validated in simple servlet-based applications such as the one in Figure 1. Hereby, the data dependencies are identified as an important non-functional, cross-cutting concern and are cleanly separated from the core functionality. The specification and policy language used however, is still in development.

Next to the presented case study, the approach of making dataflow dependencies explicit has also been validated in the component-based protocol stack framework DiPS. In DiPS [1] functional components are chained into a pipe-and-filter structure and components can share data anonymously along the pipe by means of a shared repository. Similar results were achieved within this case study.

Future work will expand the current approach to other implicit constraints and dependencies. Target tracks are the study of dataflow dependencies in Message Driven Beans, and implicit invocations in event-based systems.

References

[1] K.U.Leuven DistriNet Research Group. DiPS home page. http://www.cs.kuleuven.ac.be/cwis/research/-distrinet/projects/DIPS/.

[2] J. Hunter and W. Crawford. *Java Servlet Programming*. O'Reilly, second edition, April 2001.

[3] Walter L. Hürsch and Cristina Videira Lopes. Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA, February 1995.

[4] Java servlet technology. http://java.sun.com/products/servlet/.

[5] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[6] M. Shaw and D. Garlan. *Software Architecture - Perspectives on an emerging discipline*. Prentice-Hall, 1996.