Software Integration in Health Care

Emma Nilsson-Nyman

Department of Computer Science
Lund University

June 2007

Health care today depend on a multitude of different electronic devices. Hospitals contain everything from pure software such as record-keeping systems to medical equipment like X-ray machines.

These devices need to work together in different combinations. Combining systems of this complexity is not easy. Most of these devices are highly specialized and rarely share protocols with other devices, i.e., speak any common language. These integration problems commonly lead to customized solutions between pairs of systems. A large number of different customized solutions create an overall system which is hard to overview.

Integration is a problem in many areas. We have used a framework developed within PalCom [PP07], an ongoing research project founded by the European Union, to combine devices. This framework was developed with the aim of supporting ubiquitous computing, i.e., computing which aims to handle a future where computers are everywhere and as natural a part of our everyday life as books or road signs [Wei91]. In particular small pervasive devices such as GPS devices and digital cameras, which easily can be moved between local networks, have been studied so far within PalCom.

The system integration problems found at the hospital, and the problem of combining wireless devices in various ways, share common ground. We have studied software integration problems at the hospital through the lens of two scenarios reflecting their current workflow. We have implemented these scenarios using the PalCom framework. The first scenario deals with merging of information from several heterogeneous sources, such as several different record-keeping systems, and the second scenario deals with a future workflow for handling of lab referral forms.

One purpose of using this framework in this environment has been to evaluate how well it can handle it. The devices found at the hospital differ, somewhat, from the typical devices the framework originally was designed to handle. This new environment puts new demands on expressiveness and performance.

By studying the scenarios we have evaluated the expressiveness of the PalCom framework. Most of our work has circled around five requirements, to some extent related to Workflow Patterns [vdAHKB03], which wraps up what needs to be expressed in the studied scenarios:

- **Sequence**
  There are operations which need to be carried out in a certain order.

- **Parallel Split**
  There are points where the workflow needs to be split into parallel paths.

- **Synchronization**
  There are points where parallel workflow paths need to be synchronized into one path.

- **States**
  There is a need to express states.

- **Instances**
  There is a need to handle more than one user at the same time.

To express workflows between devices the PalCom framework uses scripts called *assemblies*. While constructing assemblies for these scenarios a need, for a graphical notation to express assembly controlled workflows, became apparent. As a result we have compared...
the assembly language to Petri nets [Pet62] and State-charts [Har87]. Based on these comparisons we suggest a graphical notation combining Petri nets with State-charts to describe the expected workflows in the scenarios.

To match up to the graphical notation we suggest corresponding language changes to the assembly language. These changes create a direct connection between the graphical notation and the assembly language.

These changes, to some extent, introduce new issues. For example, when adding support for synchronization to the assembly language questions concerning evaluation time arise. How long should the waiting time be between the point where, from a given set of events, the first event arrives until the point where the last event arrives? To deal with this issue we suggest that a time-out construct is added. This would allow a constructor of an assembly to define the evaluation time for a single event or for a group of events.

Assemblies can, besides controlling workflows, be used as user interfaces. An assembly can offer services, and based on what these services offer, a user interface can automatically be generated. In scenarios, which have states and which use these automatically generated user interfaces, the current PalCom framework is not sufficient. There need to be a way to define different states and corresponding services that should be offered in each state. We suggest changes to the language which address this problem.

In both scenarios there are a need for an assembly which can handle more than one user. In the scenarios studied, so far, in the PalCom project this has not been the case. We suggest that constructs are added which make it possible to define where state instances should be created and where instances should be thrown away.

The PalCom framework, with an assembly language modified in the way we suggest, would be much more capable of handling the type of environment found at a hospital. The visual feedback of having a graphical notation, to design workflows and states in a scenario, make it easier to implement scenarios.

References


