HAL
open science

DiaSuite: a Tool Suite To Develop Sense/Compute/Control Applications

Benjamin Bertran, Julien Bruneau, Damien Cassou, Nicolas Loriant, Emilie Balland, Charles Consel

To cite this version:

Benjamin Bertran, Julien Bruneau, Damien Cassou, Nicolas Loriant, Emilie Balland, et al.. DiaSuite: a Tool Suite To Develop Sense/Compute/Control Applications. Science of Computer Programming, 2014, Science of Computer Programming, 79, 10.1016/j.scico.2012.04.001 . hal-00702909

HAL Id: hal-00702909

https://inria.hal.science/hal-00702909v1

Submitted on 31 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DiaSuite: a Tool Suite To Develop Sense/Compute/Control Applications

Benjamin Bertran[a,1], Julien Bruneau[a,1], Damien Cassou[a,1], Nicolas Loriant[b,2], Emilie Balland[a,1], Charles Consel[a,1]

[a] *INRIA Bordeaux Sud-Ouest, 351 cours de la liberation 33405 Talence, France*
[b] *Imperial College London, South Kensington Campus, London SW7 2AZ, UK*

Abstract

We present DiaSuite, a tool suite that uses a software design approach to drive the development process. DiaSuite focuses on a specific domain, namely Sense/Compute/Control (SCC) applications. It comprises a domain-specific design language, a compiler producing a Java programming framework, a 2D-renderer to simulate an application, and a deployment framework. We have validated our tool suite on a variety of concrete applications in areas including telecommunications, building automation, robotics and avionics.

Keywords: Domain-Specific Design Language, Tool-Based Development Methodology, Generative Programming, Pervasive Computing.

1. Introduction

The *Sense/Compute/Control* architectural pattern, promoted by Taylor *et al.* [45], applies to applications that interact with an external environment. Such applications are pervasive in domains such as telecommunications, building automation, robotics and avionics. This pattern consists of *context operators* fueled by sensing *entities*. These operators refine (aggregate and interpret) the information given by the sensors. These refined data are then passed to *controller operators* that trigger actions on entities. For example, in home automation, an anti-intrusion application senses the environment using motion sensors. Then, the application uses the motion data to

[1] E-mail: first.last@inria.fr
[2] E-mail: nloriant@doc.ic.ac.uk

determine the presence of an intruder and to trigger the alarm, if necessary.

Developing SCC applications is complex because it requires to deal with a wide range of issues: heterogeneity of device capabilities, low-level device coordination, complex APIs to distributed systems technologies, *etc*. Due to their generality, existing software architecture approaches and frameworks provide limited support for the development of SCC applications. This situation forces the developer to write substantial amounts of boilerplate code to interact with devices, and glue code to interface with various underlying technologies.

Contributions. In this paper, we describe DiaSuite, a tool suite dedicated to the development of SCC applications. Specifically, we have developed a design language, dedicated to SCC applications. From such a description, a compiler produces customized support for each development stage, namely, implementation, testing, and deployment. The source code of the tools and representative applications are available on our web site.[3]

This paper goes beyond our previous publications [7, 8, 10] in that it gives a complete tour of DiaSuite and it specifically describes the rationales of its design and its implementation. In particular, we show how the application domains and the industrial case studies have largely influenced the tool development (*e.g.,* language extensions, development support). So far, we mainly reported on a development methodology dedicated to pervasive computing [8, 10] and its generalization to the SCC paradigm [7]. Recently, we have been enriching our development methodology to address non-functional concerns such as safety [33], security [24] and Quality of Service [21].

Outline. Section 2 briefly presents the development methodology underlying DiaSuite, and shows the customized support provided by DiaSuite for each development stage. Section 3 presents the application areas covered by DiaSuite to date and how they have influenced the development of the DiaSuite tools. Section 4 gives an overview of the design of the DiaSuite tools and technologies. Related work is discussed in Section 5 and conclusions are given in Section 6.

[3] `http://diasuite.inria.fr/diasuite.zip`

2. Tool-based Development Methodology

DiaSuite provides a tool-based development methodology that relies on the Sense/Compute/Control (SCC) paradigm [7] and covers the whole development process, as depicted in Figure 1. This paradigm originates from the *Sense/Compute/Control* architectural pattern [45] and is pervasively used in domains such as telecommunications, building automation, robotics and avionics. Like a programming paradigm, the SCC paradigm provides concepts and abstractions to solve a software engineering problem. These concepts and abstractions are dedicated to a design style, raising the level of abstraction above programming. Because of its dedicated nature, such a development paradigm allows a more disciplined engineering process, as advocated by Shaw [43].

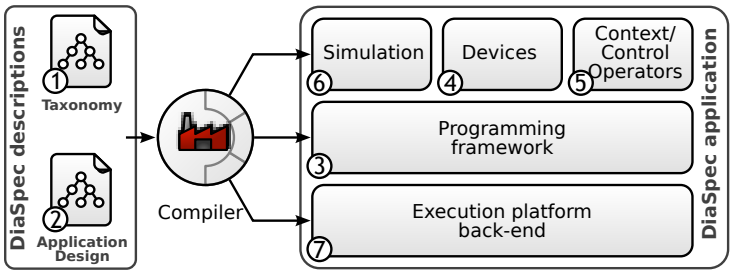

Figure 1: The development life-cycle of an SCC application using DiaSuite.

In this section, we briefly illustrate each step of the DiaSuite methodology using a simple application that displays "Hello World" on a screen when motion is detected by a motion detector. This example comes from the online tutorial available in the DiaSuite web site.[4] More details about the design language and the development methodology can be found elsewhere [7, 8, 10].

2.1. Designing

DiaSuite provides a design language, named DiaSpec, that is dedicated to the SCC paradigm. This design language consists of two layers: the taxonomy layer and the application design layer [8, 10].

[4] `http://diasuite.inria.fr`

2.1.1. Taxonomy Layer

The taxonomy layer (stage ① in Figure 1) allows a class of entities to be described. An entity is defined as a set of data sources and actuating capabilities, abstracting over devices, whether hardware or software. Additionally, attributes characterize instances of a class of entities. For the "HelloWorld" example, we only need two classes of entities: the motion sensor and the screen, as specified in Figure 2.

```
1  device LocatedDevice {
2     attribute area as String;
3  }
4
5  device MotionDetector extends LocatedDevice {
6    source motion as Boolean;
7  }
8
9  device Screen extends LocatedDevice {
10   action Display;
11 }
12
13 action Display { display(message as String); }
```

Figure 2: DiaSpec taxonomy for the "HelloWorld" application.

The `extends` keyword allows to build a hierarchy of entities. A child entity inherits the actions, sources, and attributes from the parent entity. Contrary to object-oriented programming languages, there is no notion of refinement of the inherited elements. For now, we have not seen the need for such feature in the case studies but this could be easily incorporated by relying on the Java type system.

In our example, `MotionDetector` and `Screen` both have an `area` attribute, inherited from `LocatedDevice`. This attribute allows to locate where the device is deployed (*e.g.,* the name of the room).

2.1.2. Application Design Layer

The application design layer of DiaSpec (stage ② in Figure 1) allows the application logic to be decomposed into context and control operators. The application specification can be graphically represented by its data flow using an oriented graph, whose nodes are the components, and edges indicate data exchange between components. Note that the sensors and actuators are the two facets of an entity described in the taxonomy. The data-flow graph is structured into four layers as depicted in Figure 3: (1) *Sensors* send information sensed from the environment to the context operator layer

through data *sources*, (2) *Context operators* refine (aggregate and interpret) the information given by the sensors (3) *Control operators* transform the information given by the context operators into orders for the actuators, and (4) *Actuators* trigger actions on the environment.

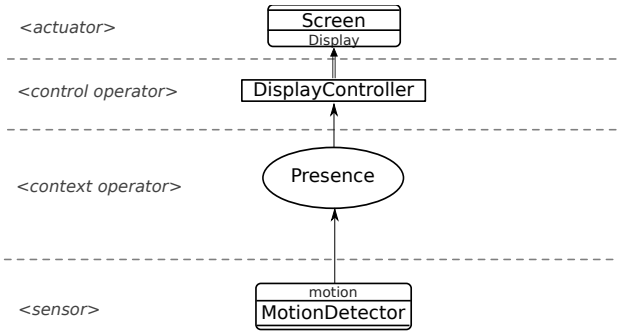

Figure 3: Graphical representation of the "HelloWorld" application design.

For example, the "HelloWorld" application is specified by a `Presence` context operator, responsible for reporting when motion is detected, and by `DisplayController` that displays messages on screens when presence is reported. The application design is given in Figure 4.

```
1 context Presence as Boolean {
2   source motion from MotionDetector;
3 }
4
5 controller DisplayController {
6   context Presence;
7   action Display on Screen;
8 }
```

Figure 4: The DiaSpec design of the "HelloWorld" application.

A DiaSpec specification is not contemplative: contrary to design documentations, DiaSpec guides and supports the remaining development stages using program generation tools.

2.2. Implementing

Given a DiaSpec description, a compiler generates a dedicated Java programming framework (stage ③ in Figure 1). This framework guides and supports the programmer to implement the various parts of the software

system: entities, context operators and control operators. Specifically, it includes an *abstract class* for each component declaration, providing *abstract methods* to guide the programming of the application logic (*e.g.,* triggering entity actions) and *concrete methods* to support the development (*e.g.,* entity discovery). Implementing a component is done by *sub-classing* the corresponding generated abstract class and by *implementing* each abstract method of the super class (stages ④ and ⑤ in Figure 1).

```
1  public class PresenceImpl extends Presence {
2
3    public PresenceImpl(ServiceConfiguration serviceConfiguration) {
4      super(serviceConfiguration);
5    }
6
7    @Override
8    protected void postInitialize() {
9      allMotionDetectors().subscribeMotion();
10   }
11
12   @Override
13   public void onNewMotion(MotionDetectorProxy proxy, Boolean motion) {
14     setPresence(motion);
15   }
16
17 }
```

Figure 5: Implementation of the `Presence` context operator.

For example, the implementation of the `Presence` context operator is defined in Figure 5. In the `postInitialize` method, the component subscribes to all the motion detectors. It would have been possible to subscribe to a subset of the motion detectors by selecting them with respect to their attributes. For example, the following expression selects motion detectors that are located in the living room:

```
motiondetectorsWhere.area(eq("LivingRoom"))
```

This entity discovery support relies on a Java-embedded, type-safe Domain-Specific Language (DSL), inspired by the fluent interfaces proposed by Fowler [19]. Existing works often use strings to express queries, deferring to runtime the detection of errors in queries. By using fluent interfaces, generated from the taxonomy declarations, the Java type checker ensures that the query is well-formed at compile time. In the expression above, it is for example not possible to pass anything else than a string to the `eq()` method as the `area` attribute is of type `String`.

The `onNewMotion` method is declared abstract in the `Presence` class and has to be implemented by the developer in the `PresenceImpl` sub-class. This method corresponds to the method logic of the `Presence` component. It is automatically called by the framework when the component is activated to process a new motion-source value published by any subscribed motion detector. For the sake of simplicity, the application logic presented here is trivial as the developer simply propagates the value from the sensor to the controller. A more realistic implementation would cross-check this information using motion detectors nearby.

The programming framework leverages the Java type checker to ensure the conformance between the design and the implementation. There are three basic conformance criteria [30]: decomposition (each component in the design is implemented), interface conformance (each component implementation conforms to its design interface), and communication integrity (each component implementation communicates only with the components it is connected to in the design). In our approach, these criteria are expressed as Java typing constraints and verified by the Java type checker during compilation. As a result, our generative approach ensures that the implementation conforms by construction to its design, in contrast to a-posteriori approaches based on a feedback-correction loop [4, 27].

2.3. Testing

Deploying an SCC application for testing purposes can be expensive and time consuming because it requires to acquire, test and configure all the necessary entities. To overcome this problem, DiaSuite includes a simulator for SCC applications, targeting a home/building environment [6] (Figure 6). This tool leverages declarations provided in the DiaSpec specification to enable a tester to instantiate simulated entities in a simulated physical environment using drag-and-drop operations. Applications are executed within the simulator without requiring any changes to the application code. This platform includes a 2D-graphical renderer, based on Siafu [31].

A tester can interact with the environment during the simulation. In particular, Siafu allows to deploy *agents* representing individuals that the tester can move within the environment. To successfully transition from a simulated environment to a real one, the DiaSuite simulator allows to combine both simulated and real entities. Hybrid simulation allows real entities to be added incrementally in the simulation, as the implementation and deployment progress.

Figure 6: DiaSuite simulator renderer.

To facilitate the development of new simulated environments, DiaSuite provides programming support.

The tester can develop simulated entities by extending the corresponding abstract class provided by the generated programming framework. In the "HelloWorld" example, we can create the `SimulatedMotionDetector` class by extending the `MotionDetector` abstract class provided by the programming framework, as shown in Figure 7. This strategy allows applications to interact with entities, regardless of their nature: simulated or real. The `SimulatedMotionDetector` class also implements the `SimulatedDevice` interface. The `receiveStimulus` method defined in this interface allows the simulated device to react to stimuli from the simulated environment.

The tester can then specify how the stimuli used by the simulated devices are produced. In the "HelloWorld" example, this interaction is encoded by specifying the behavior of the agents, as shown by Figure 8. The behavior of the agents is defined by extending the `DiaSimAgentModel` class. If an agent is in a new area, a stimulus is produced indicating her/his location. This stimulus can then be used by the simulated sensor to detect her/his presence.

```java
public class SimulatedMotionDetector extends MotionDetector
     implements SimulatedDevice {

  ...

  private static Source motionSource =
     new Source("MotionDetector", "motion", "Boolean");

  public void receiveStimulus(Stimulus stimulus) {
    if (stimulus.getSource().equals(motionSource)) {
      setMotion((Boolean) stimulus.getStimulus());
    }
  }

}
```

Figure 7: Implementation of a simulated motion detector.

```java
public class MyAgentModel extends DiaSimAgentModel {

      ...

       @Override
       public void agentMoved(Agent agent, String location) {
         Source motionSource =
              new Source("MotionDetector", "motion", "Boolean");
         publish(motionSource, true, new Location(location));
       }

}
```

Figure 8: Modeling of the agents in the simulation environment.

2.4. Deploying

An application is deployed in a specific execution platform, whether distributed (*e.g.,* RMI [16], Web Service [14], and SIP [39][5]) or local (*e.g.,* OSGi, plain Java). [6] The deployment is done in two steps. First, each component is instantiated, passing a deployment configuration to its constructor. In doing so, a component is prepared to be deployed in the target execution platform. The second step consists of initializing communication mechanisms to enable component interactions using the target platform. Associated with the Java programming framework, a Java deployment framework helps the developer select a specific execution platform (stage ⑦ in Figure 1).

Additionally, our approach allows an application to be deployed in a running platform, reusing available entities. For example, an application may be deployed on a SIP platform and leverage existing SIP entities (*e.g.,* phones and answering machines). More generally, as the entities are shared by several applications, their implementations are usually deployed separately. Then, the deployment of an application consists of only deploying the context and controller components, while reusing the available entities already deployed in the platform.

2.5. Evolution and Maintenance

To cope with evolution and maintenance, our development methodology relies on the Java type system. When the DiaSpec design is changed, a new programming framework is generated and the IDE automatically points to code locations where changes are required, as is done with any ill-typed Java programs. This approach guides the developer in applying the changes of the architect, while preserving the existing implementation. For example, an entity can be extended with additional functionalities. After regenerating the programming framework, the developer has to implement the new functionalities but do no need to adapt the rest of the code. More details about evolution of DiaSuite applications can be found elsewhere [10].

If the application is already deployed, its future evolutions can be taken into account by leveraging the versioning support offered by the runtime platform. For example, OSGi offers versioning mechanisms, allowing the update of an application without impacting the whole system.

[5] SIP stands for Session Initiation Protocol. It is a *de facto* standard for modern telephony.

[6] For now, only the plain Java and RMI back-ends are included in the public release.

3. Application Domains

DiaSuite has been used to develop applications in a broad range of domains. In this section, we present two domains, namely home/building automation and avionics. For each domain, we first instantiate the SCC paradigm with one representative example; then, we enumerate the real-size case studies we developed; finally, we discuss the impact of the domain on the design of DiaSuite. The ongoing and future domains such as robotics and assisted living are briefly discussed at the end of this section.

3.1. Home/Building Automation

Modern buildings are populated with networked equipments that are able (1) to sense the environment (*e.g.,* temperature, motion), and (2) to act on this environment (*e.g.,* trigger alarms, turn on/off lights). Applications orchestrating these equipments can clearly be designed using the SCC paradigm.

Representative Example

A representative example of the home automation domain is Heating Ventilation and Air-Conditioning (HVAC), responsible for providing thermal comfort and acceptable indoor air quality to occupants of a building. Figure 9 illustrates the DiaSpec design of an HVAC system that regulates both the temperature and the carbon dioxide for each area of a building. An occupant selects a desired temperature for a given location via a `Thermostat`. This target temperature is compared to the average temperature measured by `TemperatureSensor`s. `AirCoolerHeater`s are then requested to heat or cool, if necessary, and the thermostat display is updated accordingly. The regulation of the carbon dioxide modifies the ventilation speed depending on the level of carbon dioxide.

Case Studies

Five years ago, our research group was mainly interested in orchestrating applications in the telecommunications domain, leveraging new opportunities created by the emergence of Voice over IP (mainly based on SIP). Concurrently, a myriad of objects became networked, prompting a need to expand the scope of telecommunications beyond connecting two users. Three major projects were instrumental to explore the scope of this evolution, to devise a new approach to developing applications, and to carry out this approach with DiaSuite.

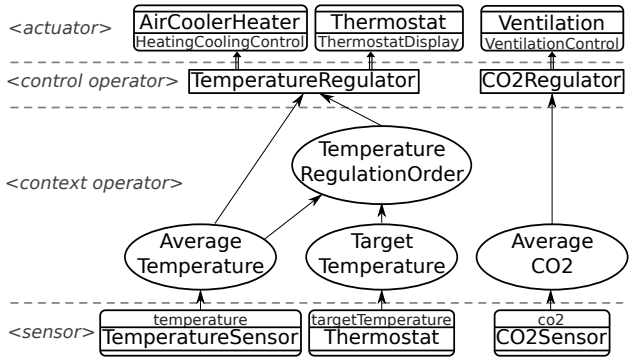

Figure 9: The HVAC system specification.

First, we collaborated with a French telecommunications company, in a two-year project, to study the convergence between VoIP and networked objects in the context of home automation. To explore the scope of this convergence, we developed a range of applications, including remote appliance control through phone keypad, TV recording via SMS, and dynamic entry phone systems [2, 3]. Second, we contributed to a two-year project named SmartImmo, which gathered major French companies in the area of building construction, installation, and management. The goal of this project was to create a service infrastructure for building automation. SmartImmo gave us the opportunity to elaborate realistic building automation scenarios (*e.g.,* parking lot management, meeting room reservation, energy monitoring) and to deploy them in a real environment.

Lastly, we worked on a project to manage a 13,500-square meters building, hosting an engineering school and its research facilities. The project led to the development of applications such as fire management, light and air conditioning management, and access control. The engineering school building was simulated using our simulation tool, allowing to assess its scalability. This simulation comprised more than 250 device instances and 300 occupants (*e.g.,* staff, researchers, students, visitors) were simultaneously simulated, testing various behavioral patterns.

Impact on the Design of DiaSuite

In a home environment, a given action such as delivering a message can be provided by a large number of devices, relying on different technologies (*e.g.,* Bluetooth, WiFi, ZigBee, Z-Wave), and using different interaction modes

(*e.g.,* spoken messages, popups). The DiaSuite notion of taxonomy allows to abstract over this heterogeneity, providing high-level entity definitions. As a result, we have managed to specify a unique taxonomy dedicated to home automation that is now used for all the applications developed in this domain. Our industrial collaborations have led us to integrate new communication protocols, such as Bluetooth and Z-Wave, and to develop new backends such as SIP. More generally, the design of the runtime has been greatly influenced by the domain of home automation. Indeed, introducing the concept of back-end libraries has allowed us to cope with the abundance of execution platforms.

Another impact of this domain on tool building is the DiaSuite simulator itself. First, the DiaSuite simulator has specifically been introduced to target home/building environments [6]. Second, large-size case studies (*e.g.,* the management of an engineering school) have led us to enrich the simulator with new features such as the automation of agent behavior and the multi-level maps to deal with the building floors. Another planned improvement is continuous models for the environment. For example, an HVAC system manipulates physical data from the environment to regulate air characteristics inside a building. In the current version of the simulator, these physical properties are simplified. For more accurate simulations, the environment should be defined by a continuous model based on differential equations. We are pursuing this line of work by defining continuous models expressed in a DSL, named Acumen [5]. This approach allows to easily connect discrete (event-based) SCC systems with continuous models of the environment.

3.2. Avionics

In avionics, an aircraft can be seen as an environment full of sensors (*e.g.,* accelerometers, gyroscopes, and GPS sensors) and actuators (*e.g.,* ailerons and elevator trim). Both critical and non-critical applications interact with this environment. For example, the entertainment system and the flight guidance system provide general flight information, from data produced by sensors, to the passengers and the pilot, respectively. Such applications can be described using the SCC paradigm.

Representative Example

Many long-haul flights are generally equipped with flight entertainment systems to watch movies, listen to music, or follow real-time flight information. Figure 10 depicts the specification of a flight entertainment application

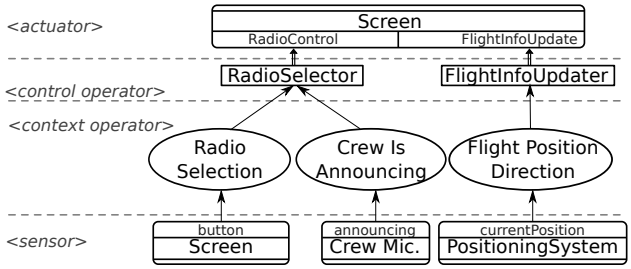

Figure 10: The flight entertainment system.

in DiaSpec. Via the screen buttons, a passenger selects an audio channel among several musical genre (*e.g.,* classical, rock). If the crew uses the microphone, all the screens switch to the *message* channel. Once the announcement is done, the application rolls back audio channels to the passenger's selections. From the aircraft positioning system, the `FlightPositionDirection` gets the current position and calculates the flight direction. The position and the direction are pushed on screens.

In a critical platform such as an aircraft, software systems have to be certified. For example, it must be demonstrated that the entertainment system gathers information from aircraft systems (*e.g.,* GPS), without any undesirable effect. Declarative approaches like DiaSuite allow to automatically verify what and how resources are used by an application.

Case Studies

Several avionics case studies have been realized in the context of a collaboration with a French airborne systems company. For example, we have developed a flight guidance application. The flight guidance application is in charge of the plane navigation and is under the supervision of the pilot. It allows the pilot to specify flight parameters (*e.g.,* the altitude) or to define a flight plan that is automatically followed. Each parameter is handled by a specific navigation mode (*e.g.,* altitude mode, heading mode). Once a mode is selected by the pilot, the flight guidance application is in charge of operating ailerons and elevators to reach the target position. For example, if the pilot specifies a heading to follow, the application compares it to the current heading, sensed by devices such as the Inertial Reference Unit, and maneuvers ailerons accordingly.

A flight guidance application has also been developed for a commercial

drone platform. The goal of this application was to make the drone autonomous by following a flight plan similar to the one in avionics.[7]

Impact on the Design of DiaSuite

The safety-critical nature of the avionics domain takes the form of stringent non-functional requirements. Applying DiaSuite to this domain led us to enrich DiaSpec with extensions to address these requirements. In doing so, we introduced declarations to allow the safety expert to specify at design time how errors are handled, guiding and facilitating the implementation of error handling code [33]. Also, we added Quality of Service (QoS) declarations [21] to express time constraints in entity/component communications. For each of these non-functional declarations, specific development support is generated. In contrast with general-purpose design driven approaches, such as UML, the SCC pattern guides the development rigorously for both functional and non-functional specifications. Ongoing work concerns the specification of fault tolerance strategies and support for existing avionics deployment technologies. To easily integrate these extensions in DiaSuite, some of the tools have been greatly refactored. For example, the runtime has been enriched with a notion of interceptors, which allows to easily add extensions without impacting the generated programming framework. For example, these interceptors are used to instrument the system with runtime guards such as QoS contracts.

In avionics, it is required to verify the behavior of the application in specific environmental conditions. Because some scenarios are difficult to create (*e.g.,* extreme flight conditions), DiaSuite has been extended with testing support that relies on a flight simulator, namely FlightGear,[8] to simulate the external environment. This work has been significantly influenced by the DiaSuite simulator for home-building environment (*e.g.,* the notion of simulated devices).

3.3. Ongoing and Future Case Studies

DiaSuite has been used in several other areas such as telecommunications [2, 3], software monitoring [7], and robotics [11]. Leveraging our expertise in avionics (*e.g.,* system criticality, fault tolerance) and home automation (*e.g.,* orchestration, system integration), we have started to apply

[7] View the drone application at `http://www.youtube.com/watch?v=9l8tRbya4vU`
[8] `http://www.flightgear.org`

our approach and tool suite to assisted living. In this domain, caregivers require applications to be tailored to meet the specific needs of end users. Yet, our approach requires domain experts, experienced architects and developers. An ongoing project is to provide end users and caregivers with a visual environment and a methodology to develop orchestration rules on top of DiaSuite, given a taxonomy of assistive building blocks [17].

DiaSuite is also a vehicle for teaching and demonstration purposes. This variety of usage contexts demonstrates the generality of DiaSuite and the robustness of its implementation. For four years, DiaSuite has been used by students of several universities for teaching (software design courses) and also in software development projects. These activities have provided us with a stream of feedback about our tools, leading to improvements ranging from syntax refinements to extended programming support. For example, the difficulties encountered by the students during the labs has motivated the development of interaction contracts to describe interactions between components [7].

A home automation system based on DiaSuite runs 24/7 in our lab. We also have demonstrated our tool suite at PerCom 2010 [9] and the simulator at MobiQuitous 2009 [6], and PerCom 2009 [25].

4. Tool Building of DiaSuite

In this section, we present the implementation of the tools and technologies forming the current public release of DiaSuite (version 2.0). In particular, we discuss several design decisions made during the implementation phase to facilitate the maintenance and evolution of DiaSuite. Figure 11 gives an overview of the relations and dependencies between DiaSuite, the external libraries and the application code.

4.1. The DiaSpec Compiler

The DiaSpec compiler is mainly composed of a parser and a programming framework generator. In the previous version of DiaSpec, the compiler relied on JastAddJ, an extensible Java compiler based on attribute grammar [18]. This choice was justified by the embedding of the syntax of DiaSpec in Java. This embedding does not exist anymore in the current version of DiaSpec. As a result, the new parser is now developed using the ANTLR parser generator [37] and the programming framework generator is based on the StringTemplate code generation template engine [37]. ANTLR is a

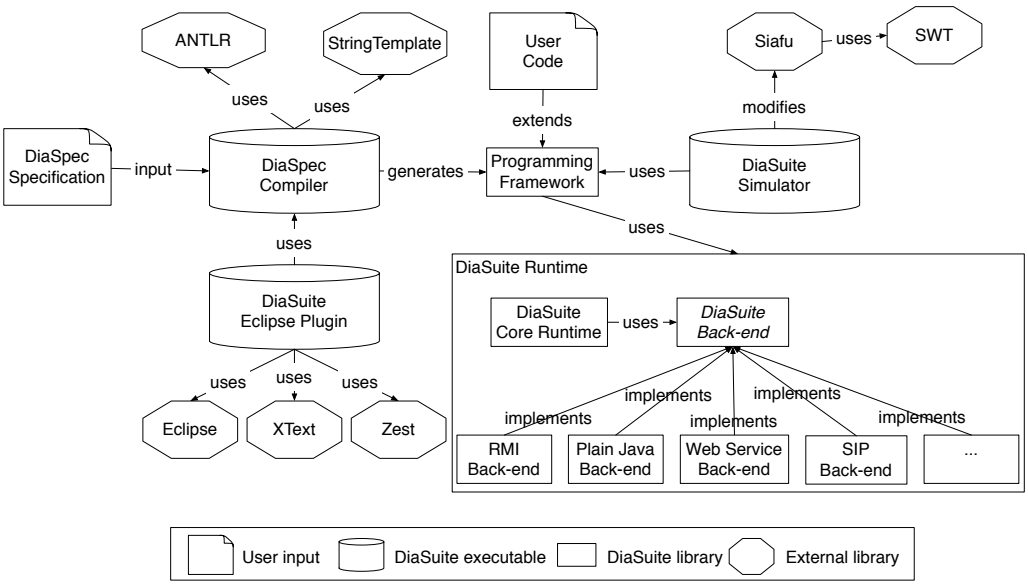

Figure 11: Tools and technologies involved in the DiaSuite architecture.

well-known parser generator that allows to generate an LL^* parser from a grammar description. StringTemplate is a Java template engine for generating formatted text output. Both tools rely on declarative domain-specific languages, facilitating the evolution and extension of the DiaSpec language.

For example, several ongoing projects in our research group are extending DiaSpec to address non-functional concerns such as safety [33], security [24] and Quality of Service (QoS) [21]. At the level of the compiler, these extensions have been developed by adding new grammar rules to the ANTLR DiaSpec specification and new StringTemplate files for the generation of the extension-specific code in the programming framework (*e.g.,* containers dedicated to QoS monitoring), minimizing the impact on the compiler's code.

4.2. The Runtime Libraries

The runtime libraries are divided in two parts: the Core Runtime library and the back-end libraries. The first part is made up of the common functionalities shared between applications and the back-ends (*e.g.,* code for entity discovery). The second part is made up of the back-end libraries, corresponding to the different execution platforms. Four back-ends are currently offered, targeting plain-Java applications, Web Services [14], SIP [39] and RMI [16].

As explained in Section 2, the execution platform is chosen at deployment time. The architecture of the Core Runtime library guarantees a clear separation between the programming framework and the back-ends, ensuring portability of the applications. When we defined the interface between the Core Runtime library and the back-ends, the main challenge was to find the suitable level of abstraction: this interface had to be generic enough to cover a broad range of execution platforms, while being concrete enough to ease the development of a new back-end. In practice, developing a new back-end consists of extending three classes:

1. The `ServiceConfiguration` class parameterizes the deployment of components (as shown in Section 2, for the deployment stage). For example, `RMIServiceConfiguration` gives information about the registry server to use.
2. `RemoteServiceInfo` contains information about component instances (*e.g.,* instance URI for the SIP back-end).
3. The `AbstractProcessor` class is the interface between the framework and the execution platform. This class implements the DiaSuite communication mechanisms (*e.g.,* publish/subscribe mechanisms, call procedures, device discovery) in terms of execution platform features (*e.g.,* SIP message forging, OSGi API calls).

Most of the back-ends rely on existing libraries. For example, the SIP back-end is based on the Java API specification for SIP signaling, namely JAIN-SIP [23].

We are now working on a new back-end for OSGi [36] that will greatly facilitate the entity life-cycle management. Furthermore, because of the design of the libraries, we are seamlessly mapping OSGi concepts into our back-end abstractions.

4.3. The DiaSuite Simulator

We studied numerous existing simulators for pervasive computing environments [6]. We decided to use Siafu [31], a 2D-graphical context simulator. This choice was motivated by two key features: (1) Siafu provides a context simulation engine to model pervasive computing environments, and (2) Siafu is written in Java and could thus be easily interfaced with our tools.

The DiaSuite simulator provides a visual editor to create simulation scenarios for SCC applications. Screenshots of this editor are presented in Figure 12. The editor allows to define layouts based on pictures for the back-

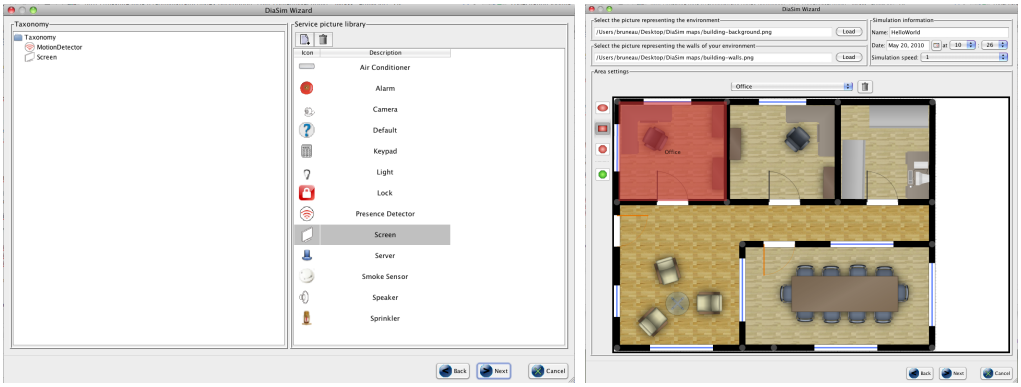

Figure 12: DiaSuite simulator editor.

ground and walls. Based on the DiaSpec specification, the editor allows to add simulated devices and agents to a layout. When saving the layout, dedicated simulation support is automatically generated, providing the tester with a simulation programming framework to develop simulated entities.

Siafu-specific code is generated in the simulation support to take into account the simulated environment layout (*e.g.,* agent instantiation, stimulus emission and reception). Thus, the tester can launch his simulation in Siafu without knowing Siafu specificities. The simulation support provides all the necessary glue code to make DiaSuite applications interact with Siafu.

4.4. The Ant Tasks Support

DiaSuite is provided with an Ant file (`buildScenario.xml`) that assists the developer in achieving the main tasks towards producing a DiaSuite application. This script file allows to automate processes such as code generation and system configuration, without depending on a particular IDE.

Executing the `installScenario` task initializes the DiaSuite project. This action creates the directories and libraries needed to start the implementation of a DiaSuite application. This step is a prerequisite to the development of the DiaSpec specification.

From a DiaSpec specification, the `framework` task allows to generate the programming framework, calling the DiaSpec compiler. Once the programming framework is generated, the developer can implement the entities, as well as the context and control operators, by extending the abstract classes of the framework.

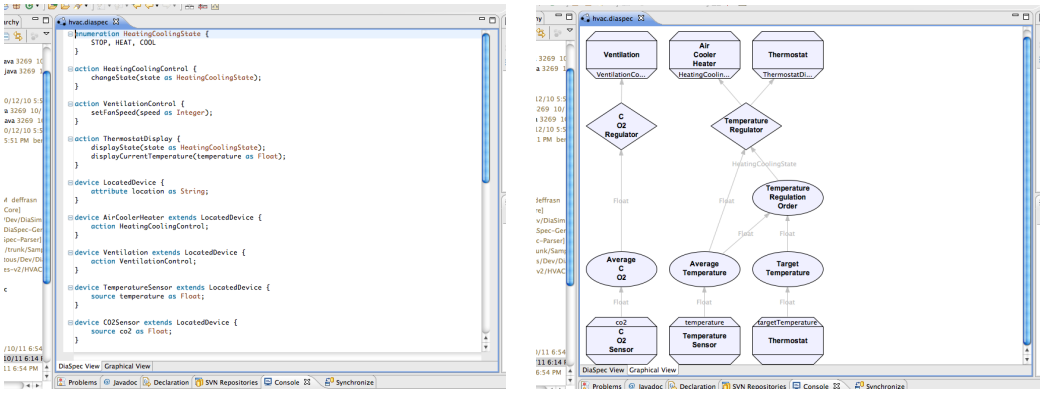

Figure 13: DiaSpec Eclipse editor plugin.

Finally, there are two specific tasks for the simulation. The `launchEdition` task allows to launch the editor of simulated environments (Figure 12). When saving the layout, the simulation support is automatically generated. Then, the user can implement the behavior of the simulated entities and launch the simulation using the `launchSimulation` task.

4.5. The Eclipse Plugin

Our DiaSpec plugin for Eclipse eases the editing of DiaSpec specifications. It provides standard tools for textual languages, such as syntax highlighting, code completion, and code snippets. The plugin also generates a graphical view of a specification. This view allows to quickly understand a specification and provides additional support to developers. Figure 13 shows the DiaSpec Eclipse plugin in action. This plugin is based on XText,[9] a framework for language development, and Zest,[10] a toolkit to draw the graphical specification dataflow.

The XText grammar is very similar to the one of ANTLR. Despite few differences, we have to maintain both grammars. It would be interesting to develop a converter between XText and ANTLR to reduce this maintenance cost.

[9] `http://www.eclipse.org/Xtext`
[10] `http://www.eclipse.org/gef/zest`

5. Related Work

In this section, we compare DiaSuite with related development tools. First, we present generalist tools such as Model-Driven Engineering (MDE) or Architecture Description Languages (ADL). Then, we present domain-specific tools such as context management middlewares or simulators dedicated to pervasive computing. Contrary to DiaSuite, most of these tools focus on a particular stage of the development process.

Model-Driven Engineering. Model-Driven Engineering (MDE) uses models and model transformations as a way to specify software architectures and implementations [41]. The goal of these approaches is to raise the level of abstraction in program specifications and generate a working implementation from such a specification. MDE has been used to develop SCC applications. For example, PervML [42] proposes a conceptual framework suitable for the specification of context-aware pervasive computing systems. This conceptual framework relies on UML diagrams to model pervasive computing concerns. Even though PervML proposes a conceptual framework for developing pervasive computing systems, it only provides the user with generic tools. Thus, tailoring these generic tools to fit the pervasive computing domain incurs an overhead, compared to the DiaSuite approach.

Architecture Description Languages. Archface [46] is the work that is the most similar to our approach. It proposes an interface between design and code: Archface is both a general-purpose ADL and a programming-level interface. In our approach, being domain specific allows domain experts to design their architecture without mixing programming concepts, while enabling the generation of dedicated support in the implementation. For example, the DiaSpec compiler generates dedicated programming support to discover entities based on the taxonomy definition.

Context management middlewares. Schmidt *et al.* [40], Chen and Kotz [12], and Dey *et al.* [15], have proposed middleware layers to acquire and process context information from sensors. They also include context management support that maintains repositories of context information and provides rich query facilities to context-aware applications [26, 29]. This special-purpose support enables to factor most aspects of context management out of the application code. Henricksen *et al.* take this approach one step further by introducing a language to model the computation of context information [22, 32]. These context management systems focus on programming

support for acquiring and processing context information. In contrast, the DiaSuite approach provides support for each development stage, namely, design, implementation, testing, and deployment.

Programming frameworks. Many software layers have been proposed to ease the burden of SCC developers [20, 38]. These software layers attempt to cover as much of the SCC domain as possible in a single programming framework. Such a strategy often leads to large APIs, which provide little guidance to the developer and require boilerplate code to customize the middleware to the characteristics of the application area. On the contrary, a DiaSuite-generated programming framework specifically targets a given application, narrowing the developer's usage of an API to methods of interest.

Simulators. Few simulators are dedicated to the testing of pervasive computing applications [1, 34, 35, 44]. The Stage and Gazebo simulators are dedicated to the Player robot framework [13] and have been applied to the pervasive computing domain [28]. However, users of these simulators have to manually specialize the simulator for every new application area. In contrast, the DiaSuite simulator relies on the DiaSpec descriptions to automatically customize the simulation tools (*i.e.,* the editor and the renderer).

6. Conclusion

In this paper, we have given a tour of DiaSuite, a development environment dedicated to SCC applications. This development environment relies on a domain-specific design language named DiaSpec that allows to specify an SCC application using a dedicated architectural pattern. From this specification, DiaSuite provides customized support for each development phase, namely implementation, testing, and deployment. DiaSuite has been validated in a wide range of application areas. In this paper, we have shown the application areas that have been covered to date.

DiaSuite is being extended in various directions. For example, interaction contracts have been recently introduced to describe interactions between components, allowing extensive programming support to be generated and new static verifications to be performed [7]. Another avenue of research is to enrich DiaSpec with non-functional concerns, such as safety [33], security [24] and Quality of Service [21], leveraging further our declarative approach to software design.

References

[1] John J. Barton and Vikram Vijayaraghavan. UBIWISE, a ubiquitous wireless infrastructure simulation environment. Technical report, Hewlett Packard, 2002.

[2] Benjamin Bertran, Charles Consel, Wilfried Jouve, Hongyu Guan, and Patrice Kadionik. SIP as a universal communication bus: A methodology and an experimental study. In *ICC'10: Proceedings of the IEEE International Conference on Communications*, 2010.

[3] Benjamin Bertran, Charles Consel, Patrice Kadionik, and Bastien Lamer. A SIP-based home automation platform: An experimental study. In *ICIN'09: Proceedings of the 13th International Conference on Intelligence in Next Generation Networks*, 2009.

[4] Johan Brichau, Andy Kellens, Sergio Castro, and Theo D'Hondt. Enforcing structural regularities in software using IntensiVE. *Science of Computer Programming*, 75(4):232 – 246, 2010.

[5] Julien Bruneau, Charles Consel, Marcia O'Malley, Walid Taha, and Wail Masry Hannourah. Preliminary results in virtual testing for smart buildings (poster). In *MobiQuitous'10: Proceedings of the 7th International ICST Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, 2010.

[6] Julien Bruneau, Wilfried Jouve, and Charles Consel. DiaSim: A parameterized simulator for pervasive computing applications (demonstration). In *MobiQuitous'09: Proceedings of the 6th International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 1–10. IEEE Computer Society, 2009.

[7] Damien Cassou, Emilie Balland, Charles Consel, and Julia Lawall. Leveraging software architectures to guide and verify the development of Sense/Compute/Control applications. In *ICSE'11: Proceedings of the 33rd International Conference on Software Engineering*, pages 431–440. ACM, 2011.

[8] Damien Cassou, Benjamin Bertran, Nicolas Loriant, and Charles Consel. A generative programming approach to developing pervasive computing

systems. In *GPCE'09: Proceedings of the 8th International Conference on Generative Programming and Component Engineering*, pages 137–146. ACM, 2009.

[9] Damien Cassou, Julien Bruneau, and Charles Consel. A tool suite to prototype pervasive computing applications (demonstration). In *PerCom'10: Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications*, pages 820–822. IEEE Computer Society, 2010.

[10] Damien Cassou, Julien Bruneau, Charles Consel, and Emilie Balland. Towards a tool-based development methodology for pervasive computing applications. *IEEE Transactions on Software Engineering*, 99, 2011.

[11] Damien Cassou, Serge Stinckwich, and Pierrick Koch. Using the DiaSpec design language and compiler to develop robotics systems. In *DSLRob'11: Proceedings of the 2nd International Workshop on Domain-Specific Languages and models for ROBotic systems*, San Francisco, CA, USA, September 2011.

[12] Guanling Chen and David Kotz. Context aggregation and dissemination in ubiquitous computing systems. In *WMCSA'02: Proceedings of the 4th Workshop on Mobile Computing Systems and Applications*, pages 105–114, Washington, DC, USA, 2002. IEEE Computer Society.

[13] Toby H.J. Collett, Bruce A. MacDonald, and Brian P. Gerkey. Player 2.0: Toward a practical robot programming framework. In *ACRA'05: Proceedings of the 7th Australasian Conference on Robotics and Automation*, pages 1–9, Sydney, Australia, 2005.

[14] World Wide Web Consortium. Web services architecture, 2004. `http://www.w3.org/TR/ws-arch`.

[15] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, 2001.

[16] Troy Bryan Downing. *Java RMI: Remote Method Invocation*. IDG Books Worldwide, Inc., Foster City, CA, USA, 1998.

[17] Zoé Drey, Julien Mercadal, and Charles Consel. A taxonomy-driven approach to visually prototyping pervasive computing applications. In *DSL WC'09: Proceedings of the 1st Working Conference on Domain-Specific Languages*, volume 5658, pages 78–99, 2009.

[18] Torbjörn Ekman and Görel Hedin. The JastAdd extensible java compiler. In *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'07*, pages 773–774. ACM, 2007.

[19] Martin Fowler, 2005. `http://www.martinfowler.com/bliki/FluentInterface.html`.

[20] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste. Project AURA: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1:22–31, 2002.

[21] Stéphanie Gatti, Emilie Balland, and Charles Consel. A step-wise approach for integrating QoS throughout software development. In *FASE'11: Proceedings of the 14th European Conference on Fundamental Approaches to Software Engineering*, volume 6603 of *LNCS*, pages 217–231. Springer, 2011.

[22] Karen Henricksen and Jadwiga Indulska. A software engineering framework for context-aware pervasive computing. In *PerCom'04: Proceedings of the 2nd International Conference on Pervasive Computing and Communications*, pages 77–86. IEEE Computer Society, 2004.

[23] JAIN-SIP. `https://jain-sip.dev.java.net`.

[24] Henner Jakob, Charles Consel, and Nicolas Loriant. Architecturing conflict handling of pervasive computing resources. In *DAIS'11: Proceedings of the 11th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume 6723 of *LNCS*, pages 92–105. Springer, 2011.

[25] Wilfried Jouve, Julien Bruneau, and Charles Consel. DiaSim: A parameterized simulator for pervasive computing applications (demonstration). In *PerCom'09: Proceedings of the 7th IEEE International Conference on Pervasive Computing and Communications*, pages 1–3. IEEE Computer Society, 2009.

[26] Glenn Judd and Peter Steenkiste. Providing contextual information to pervasive computing applications. In *PerCom'03: Proceedings of the 1st International Conference on Pervasive Computing and Communications*, pages 133–142. IEEE Computer Society, 2003.

[27] Foutse Khomh, Stephane Vaucher, Yann Gaël Guéhéneuc, and Houari Sahraoui. BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. *Journal of Systems and Software*, 84(4):559–572, 2011.

[28] Matthias Kranz, Radu Bogdan Rusu, Alexis Maldonado, Michael Beetz, and Albrecht Schmidt. A Player/Stage system for context-aware intelligent environments. In *UbiSys'06: Proceedings of the System Support for Ubiquitous Computing Workshop*, pages 1–7, 2006.

[29] Hui Lei, Daby M. Sow, John S. Davis, II, Guruduth Banavar, and Maria R. Ebling. The design and applications of a context service. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):45–55, 2002.

[30] D.C. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9), 1995.

[31] Miquel Martin and Petteri Nurmi. A generic large scale simulator for ubiquitous computing (poster). In *MobiQuitous'06: Proceedings of the 3rd International Conference on Mobile and Ubiquitous Systems: Networking & Services*, pages 1–3, San Jose, CA, USA, 2006. IEEE Computer Society.

[32] Ted McFadden, Karen Henricksen, Jadwiga Indulska, and Peter Mascaro. Applying a disciplined approach to the development of a context-aware communication application. In *PerCom'05: Proceedings of the 3rd International Conference on Pervasive Computing and Communications*, pages 300–306. IEEE Computer Society, 2005.

[33] Julien Mercadal, Quentin Enard, Charles Consel, and Nicolas Loriant. A domain-specific approach to architecturing error handling in pervasive computing. In *OOPSLA'10: Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 47–61. ACM, 2010.

[34] Ricardo Morla and Nigel Davies. Evaluating a location-based application: A hybrid test and simulation environment. *IEEE Pervasive Computing*, 3(3):48–56, 2004.

[35] Eleanor O'Neill, Martin Klepal, David Lewis, Tony O'Donnell, Declan O'Sullivan, and Dirk Pesch. A testbed for evaluating human interaction with ubiquitous computing environments. In *TridentCom'05: Proceedings of the 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 60–69. IEEE Computer Society, 2005.

[36] OSGi Alliance. `http://www.osgi.org`.

[37] Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.

[38] Anand Ranganathan, Shiva Chetan, Jalal Al-Muhtadi, Roy H. Campbell, and M. Dennis Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *PerCom'05: Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, pages 7–16. IEEE Computer Society, 2005.

[39] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. SIP: Session Initiation Protocol. Technical report, RFC 3261, 2002. `http://www.ietf.org/rfc/rfc3261.txt`.

[40] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *HUC'99: Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 89–101. Springer, 1999.

[41] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, 2006.

[42] Estefanía Serral, Pedro Valderas, and Vicente Pelechano. Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, 6:254–280, April 2010.

[43] Mary Shaw. Beyond objects: A software design paradigm based on process control. *SIGSOFT Software Engineering Notes*, 20:27–38, January 1995.

[44] Sameer Sundresh, Wooyoung Kim, and Gul Agha. SENS: A sensor, environment and network simulator. In *Proceedings of the 37th Annual Simulation Symposium*, pages 221–230, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[45] Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[46] Naoyasu Ubayashi, Jun Nomura, and Tetsuo Tamai. Archface: A contract place where architectural design and code meet together. In *ICSE'10: Proceedings of the 32nd International Conference on Software Engineering*, pages 75–84, New York, NY, USA, 2010. ACM.