## How a Project Starts

**The scene:** Meeting room at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

**The players:** Mal Golden, senior manager, product development; Lisa Perez, marketing manager; Lee Warren, engineering manager; Joe Camalleri, executive VP, business development

**The conversation:**

**Joe:** Okay, Lee, what's this I hear about your folks developing a what? A generic universal wireless box?

**Lee:** It's pretty cool . . . about the size of a small matchbook . . . we can attach it to sensors of all kinds, a digital camera, just about anything. Using the 802.11g wireless protocol. It allows us to access the device's output without wires. We think it'll lead to a whole new generation of products.

**Joe:** You agree, Mal?

**Mal:** I do. In fact, with sales as flat as they've been this year, we need something new. Lisa and I have been doing a little market research, and we think we've got a line of products that could be big.

**Joe:** How big . . . bottom line big?

**Mal (avoiding a direct commitment):** Tell him about our idea, Lisa.

**Lisa:** It's a whole new generation of what we call "home management products." We call 'em *SafeHome*. They use the new wireless interface, provide homeowners or small-business people with a system that's controlled by their PC—home security, home surveillance, appliance and device control—you know, turn down the home air conditioner while you're driving home, that sort of thing.

**Lee (jumping in):** Engineering's done a technical feasibility study of this idea, Joe. It's doable at low manufacturing cost. Most hardware is off-the-shelf. Software is an issue, but it's nothing that we can't do.

**Joe:** Interesting. Now, I asked about the bottom line.

**Mal:** PCs have penetrated over 70 percent of all households in the USA. If we could price this thing right, it could be a killer-App. Nobody else has our wireless box . . . it's proprietary. We'll have a 2-year jump on the competition. Revenue? Maybe as much as 30 to 40 million dollars in the second year.

**Joe (smiling):** Let's take this to the next level. I'm interested.

## SafeHome

### *Selecting a Process Model, Part 1*

**The scene:** Meeting room for the software engineering group at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.

**The players:** Lee Warren, engineering manager; Doug Miller, software engineering manager; Jamie Lazar, software team member; Vinod Raman, software team member; and Ed Robbins, software team member.

**The conversation:**

**Lee:** So let's recapitulate. I've spent some time discussing the *SafeHome* product line as we see it at the moment. No doubt, we've got a lot of work to do to simply define the thing, but I'd like you guys to begin thinking about how you're going to approach the software part of this project.

**Doug:** Seems like we've been pretty disorganized in our approach to software in the past.

**Ed:** I don't know, Doug, we always got product out the door.

**Doug:** True, but not without a lot of grief, and this project looks like it's bigger and more complex than anything we've done in the past.

**Jamie:** Doesn't look that hard, but I agree . . . our ad hoc approach to past projects won't work here, particularly if we have a very tight time line.

**Doug (smiling):** I want to be a bit more professional in our approach. I went to a short course last week and learned a lot about software engineering . . . good stuff. We need a process here.

**Jamie (with a frown):** My job is to build computer programs, not push paper around.

**Doug:** Give it a chance before you go negative on me. Here's what I mean. [Doug proceeds to describe the process framework described in this chapter and the prescriptive process models presented to this point.]

**Doug:** So anyway, it seems to me that a linear model is not for us . . . assumes we have all requirements up front and, knowing this place, that's not likely.

**Vinod:** Yeah, and it sounds way too IT-oriented . . . probably good for building an inventory control system or something, but it's just not right for *SafeHome*.

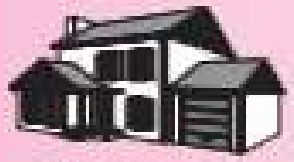
**Doug:** I agree.

**Ed:** That prototyping approach seems OK. A lot like what we do here anyway.

**Vinod:** That's a problem. I'm worried that it doesn't provide us with enough structure.

**Doug:** Not to worry. We've got plenty of other options, and I want you guys to pick what's best for the team and best for the project.

## SafeHome



### *Selecting a Process Model, Part 2*

**The scene:** Meeting room for the software engineering group at CPI Corporation, a company that makes consumer products for home and commercial use.

**The players:** Lee Warren, engineering manager; Doug Miller, software engineering manager; Vinod and Jamie, members of the software engineering team.

**The conversation:** [Doug describes evolutionary process options.]

**Jamie:** Now I see something I like. An incremental approach makes sense, and I really like the flow of that spiral model thing. That's keepin' it real.

**Vinod:** I agree. We deliver an increment, learn from customer feedback, replan, and then deliver another increment. It also fits into the nature of the product. We can have something on the market fast and then add functionality with each version, er, increment.

**Lee:** Wait a minute. Did you say that we regenerate the plan with each tour around the spiral, Doug? That's not so great; we need one plan, one schedule, and we've got to stick to it.

**Doug:** That's old-school thinking, Lee. Like the guys said, we've got to keep it real. I submit that it's better to tweak the plan as we learn more and as changes are requested. It's way more realistic. What's the point of a plan if it doesn't reflect reality?

**Lee** (frowning): I suppose so, but . . . senior management's not going to like this . . . they want a fixed plan.

**Doug** (smiling): Then you'll have to reeducate them, buddy.

## SafeHome

### *Considering Agile Software Development*

**The scene:** Doug Miller's office.

**The Players:** Doug Miller, software engineering manager; Jamie Lazar, software team member; Vinod Raman, software team member.

**The conversation:**

(A knock on the door, Jamie and Vinod enter Doug's office)

**Jamie:** Doug, you got a minute?

**Doug:** Sure Jamie, what's up?

**Jamie:** We've been thinking about our process discussion yesterday . . . you know, what process we're going to choose for this new *SafeHome* project.

**Doug:** And?

**Vinod:** I was talking to a friend at another company, and he was telling me about Extreme Programming. It's an agile process model . . . heard of it?

**Doug:** Yeah, some good, some bad.

**Jamie:** Well, it sounds pretty good to us. Lets you develop software really fast, uses something called pair programming to do real-time quality checks . . . it's pretty cool, I think.

**Doug:** It does have a lot of really good ideas. I like the pair-programming concept, for instance, and the idea that stakeholders should be part of the team.

**Jamie:** Huh? You mean that marketing will work on the project team with us?

**Doug (nodding):** They're a stakeholder, aren't they?

**Jamie:** Jeez . . . they'll be requesting changes every five minutes.

**Vinod:** Not necessarily. My friend said that there are ways to "embrace" changes during an XP project.

**Doug:** So you guys think we should use XP?

**Jamie:** It's definitely worth considering.

**Doug:** I agree. And even if we choose an incremental model as our approach, there's no reason why we can't incorporate much of what XP has to offer.

**Vinod:** Doug, before you said "some good, some bad." What was the "bad"?

**Doug:** The thing I don't like is the way XP downplays analysis and design . . . sort of says that writing code is where the action is

(The team members look at one another and smile.)

**Doug:** So you agree with the XP approach?

**Jamie (speaking for both):** Writing code is what we do, Boss!

**Doug (laughing):** True, but I'd like to see you spend a little less time coding and then recoding and a little more time analyzing what has to be done and designing a solution that works.

**Vinod:** Maybe we can have it both ways, agility with a little discipline.

**Doug:** I think we can, Vinod. In fact, I'm sure of it.

## *Communication Mistakes*

**The scene:** Software engineering team workspace

**The players:** Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member.

**The conversation:**

**Ed:** "What have you heard about this *SafeHome* project?"

**Vinod:** "The kick-off meeting is scheduled for next week."

**Jamie:** "I've already done a little bit of investigation, but it didn't go well."

**Ed:** "What do you mean?"

**Jamie:** "Well, I gave Lisa Perez a call. She's the marketing honcho on this thing."

**Vinod:** "And . . . ?"

**Jamie:** "I wanted her to tell me about *SafeHome* features and functions . . . that sort of thing. Instead, she began asking me questions about security systems, surveillance systems . . . I'm no expert."

**Vinod:** "What does that tell you?"

(Jamie shrugs.)

**Vinod:** "That marketing will need us to act as consultants and that we'd better do some homework on this product area before our kick-off meeting. Doug said that he wanted us to 'collaborate' with our customer, so we'd better learn how to do that."

**Ed:** "Probably would have been better to stop by her office. Phone calls just don't work as well for this sort of thing."

**Jamie:** "You're both right. We've got to get our act together or our early communications will be a struggle."

**Vinod:** "I saw Doug reading a book on 'requirements engineering.' I'll bet that lists some principles of good communication. I'm going to borrow it from him."

**Jamie:** "Good idea . . . then you can teach us."

**Vinod (smiling):** "Yeah, right."

## *Conducting a Requirements Gathering Meeting*

**The scene:** A meeting room. The first requirements gathering meeting is in progress.

**The players:** Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

**The conversation:**

**Facilitator (pointing at whiteboard):** So that's the current list of objects and services for the home security function.

**Marketing person:** That about covers it from our point of view.

**Vinod:** Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

**Marketing person:** Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

**Facilitator:** Does that also add some constraints?

**Jamie:** It does, both technical and legal.

**Production rep:** Meaning?

**Jamie:** We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

**Doug:** Very true.

**Marketing:** But we still need that . . . just be sure to stop an outsider from getting in.

**Ed:** That's easier said than done and . . .

**Facilitator (interrupting):** I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

**Facilitator:** I have a feeling there's still more to consider here.

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

**SAFEHOME**

## *Developing a Preliminary User Scenario*

**The scene:** A meeting room, continuing the first requirements gathering meeting.

**The players:** Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

**The conversation:**

**Facilitator:** We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

**Jamie:** How?

**Facilitator:** We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

**Jamie:** What if I forget my password?

**Facilitator (interrupting):** Good point, Jamie, but let's not address that now. We'll make a note of that and call it an *exception*. I'm sure there'll be others.

**Marketing person:** After I enter the passwords, a screen representing all *SafeHome* functions will appear. I'd select the home security function. The system might request that I verify who I am, say, by asking for my address or phone number or something. It would then display a picture of the security system control panel along with a list of functions that I can perform—arm the system, disarm the system, disarm one or more sensors. I suppose it might also allow me to reconfigure security zones and other things like that, but I'm not sure.

**Marketing person:** Um . . . well, this is the kind of thing I'd do if I was away from home and I had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

**Facilitator (smiling):** That's the reason you'd do it . . . tell me how you'd actually do this.

**Marketing person:** Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user id and . . .

**Vinod (interrupting):** The Web page would have to be secure, encrypted, to guarantee that we're safe and . . .

**Facilitator (interrupting):** That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK?

**Vinod:** No problem.

**Marketing person:** So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords.

(As the marketing person continues talking, Doug takes copious notes; these form the basis for the first informal usage scenario. Alternatively, the marketing person could have been asked to write the scenario, but this would be done outside the meeting.)

## SafeHome

### *Developing a High-Level Use-Case Diagram*

**The scene:** A meeting room, continuing the requirements gathering meeting

**The players:** Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

**The conversation:**

**Facilitator:** We've spent a fair amount of time talking about *SafeHome* home security functionality. During the break I sketched a use case diagram to summarize the important scenarios that are part of this function. Take a look.

(All attendees look at Figure 5.2.)

**Jamie:** I'm just beginning to learn UML notation.[14] So the home security function is represented by the big box with the ovals inside it? And the ovals represent use cases that we've written in text?

**Facilitator:** Yep. And the stick figures represent actors—the people or things that interact with the system as described by the use case . . . oh, I use the labeled square to represent an actor that's not a person . . . in this case, sensors.

**Doug:** Is that legal in UML?

**Facilitator:** Legality isn't the issue. The point is to communicate information. I view the use of a humanlike stick figure for representing a device to be misleading. So I've adapted things a bit. I don't think it creates a problem.

**Vinod:** Okay, so we have use-case narratives for each of the ovals. Do we need to develop the more detailed template-based narratives I've read about?

**Facilitator:** Probably, but that can wait until we've considered other *SafeHome* functions.

**Marketing person:** Wait, I've been looking at this diagram and all of a sudden I realize we missed something.

**Facilitator:** Oh really. Tell me what we've missed.

(The meeting continues.)

## SafeHome

### *Preliminary Behavioral Modeling*

**The scene:** A meeting room, continuing the requirements meeting.

**The players:** Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

**The conversation:**

**Facilitator:** We've just about finished talking about *SafeHome* home security functionality. But before we do, I want to discuss the behavior of the function.

**Marketing person:** I don't understand what you mean by behavior.

**Ed (smiling):** That's when you give the product a "timeout" if it misbehaves.

**Facilitator:** Not exactly. Let me explain.

(The facilitator explains the basics of behavioral modeling to the requirements gathering team.)

**Marketing person:** This seems a little technical. I'm not sure I can help here.

**Facilitator:** Sure you can. What behavior do you observe from the user's point of view?

**Marketing person:** Uh . . . well, the system will be *monitoring* the sensors. It'll be *reading commands* from the homeowner. It'll be *displaying* its status.

**Facilitator:** See, you can do it.

**Jamie:** It'll also be *polling* the PC to determine if there is any input from it, for example, Internet-based access or configuration information.

**Vinod:** Yeah, in fact, *configuring the system* is a state in its own right.

**Doug:** You guys are rolling. Let's give this a bit more thought . . . is there a way to diagram this stuff?

**Facilitator:** There is, but let's postpone that until after the meeting.

## *The Start of a Negotiation*

**The scene:** Lisa Perez's office, after the first requirements gathering meeting.

**The players:** Doug Miller, software engineering manager and Lisa Perez, marketing manager.

**Lisa (smiling):** Yeah, they actually told me they got into it and it wasn't a "propeller head activity."

**Doug (laughing):** I'll be sure to take off my techie beanie the next time I visit . . . Look, Lisa, I think we may have a problem with getting all of the functionality for the home security system out by the dates your management is talking about. It's early, I know, but I've already been doing a little back-of-the-envelope planning and . . .

**Lisa (frowning):** We've got to have it by that date, Doug. What functionality are you talking about?

**Doug:** I figure we can get full home security functionality out by the drop-dead date, but we'll have to delay Internet access 'til the second release.

**Lisa:** Doug, it's the Internet access that gives *SafeHome* "gee whiz" appeal. We're going to build our entire marketing campaign around it. We've gotta have it!

**The conversation:**

**Lisa:** So, I hear the first meeting went really well.

**Doug:** Actually, it did. You sent some good people to the meeting . . . they really contributed.

**Doug:** I understand your situation, I really do. The problem is that in order to give you Internet access, we'll have to have a fully secure website up and running. That takes time and people. We'll also have to build a lot of additional functionality into the first release . . . I don't think we can do it with the resources we've got.

**Lisa (still frowning):** I see, but you've got to figure out a way to get it done. It's pivotal to home security functions and to other functions as well . . . those can wait until the next releases . . . I'll agree to that.

Lisa and Doug appear to be at an impasse, and yet they must negotiate a solution to this problem. Can they both "win" here? Playing the role of a mediator, what would you suggest?

## *Domain Analysis*

**The scene:** Doug Miller's office, after a meeting with marketing.

**The players:** Doug Miller, software engineering manager, and Vinod Raman, a member of the software engineering team.

**The conversation:**

**Doug:** I need you for a special project, Vinod. I'm going to pull you out of the requirements gathering meetings.

**Vinod (frowning):** Too bad. That format actually works . . . I was getting something out of it. What's up?

**Doug:** Jamie and Ed will cover for you. Anyway, marketing insists that we deliver the Internet capability along with the home security function in the first release of *SafeHome*. We're under the gun on this . . . not enough time or people, so we've got to solve both problems—the PC interface and the Web interface—at once.

**Vinod (looking confused):** I didn't know the plan was set . . . we're not even finished with requirements gathering.

**Doug (a wan smile):** I know, but the time lines are so short that I decided to begin strategizing with marketing right now . . . anyhow, we'll revisit any tentative plan once we have the info from all of the requirements gathering meetings.

**Vinod:** Okay, what's up? What do you want me to do?

**Doug:** Do you know what "domain analysis" is?

**Vinod:** Sort of. You look for similar patterns in Apps that do the same kinds of things as the App you're building. If possible, you then steal the patterns and reuse them in your work.

**Doug:** Not sure I like the word *steal*, but basically you have it right. What I'd like you to do is to begin researching existing user interfaces for systems that control something like *SafeHome*. I want you to propose a set of patterns and analysis classes that can be common to both the PC-based interface that'll sit in the house and the browser-based interface that is accessible via the Internet.

**Vinod:** We can save time by making them the same . . . why don't we just do that?

**Doug:** Ah . . . it's nice to have people who think like you do. That's the whole point—we can save time and effort if both interfaces are nearly identical, implemented with the same code, blah, blah, that marketing insists on.

**Vinod:** So you want, what—classes, analysis patterns, design patterns?

**Doug:** All of 'em. Nothing formal at this point. I just want to get a head start on our internal analysis and design work.

**Vinod:** I'll go to our class library and see what we've got. I'll also use a patterns template I saw in a book I was reading a few months back.

**Doug:** Good. Go to work.

## SafeHome

### *Developing Another Preliminary User Scenario*

**The scene:** A meeting room, during the second requirements gathering meeting.

**The players:** Jamie Lazar, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

**The conversation:**

**Facilitator:** It's time that we begin talking about the *SafeHome* surveillance function. Let's develop a user scenario for access to the surveillance function.

**Jamie:** Who plays the role of the actor on this?

**Facilitator:** I think Meredith (a marketing person) has been working on that functionality. Why don't you play the role?

**Meredith:** You want to do it the same way we did it last time, right?

**Facilitator:** Right . . . same way.

**Meredith:** Well, obviously the reason for surveillance is to allow the homeowner to check out the house while he or she is away, to record and play back video that is captured . . . that sort of thing.

**Ed:** Will we use compression to store the video?

**Facilitator:** Good question, Ed, but let's postpone implementation issues for now. Meredith?

**Meredith:** Okay, so basically there are two parts to the surveillance function . . . the first configures the system including laying out a floor plan—we have to have tools to help the homeowner do this—and the second part is the actual surveillance function itself. Since the layout is part of the configuration activity, I'll focus on the surveillance function.

**Facilitator (smiling):** Took the words right out of my mouth.

**Meredith:** Um . . . I want to gain access to the surveillance function either via the PC or via the Internet. My feeling is that the Internet access would be more frequently used. Anyway, I want to be able to display camera views on a PC and control pan and zoom for a specific camera. I specify the camera by selecting it from the house floor plan. I want to selectively record camera output and replay camera output. I also want to be able to block access to one or more cameras with a specific password. I also want the option of seeing small windows that show views from all cameras and then be able to pick the one I want enlarged.

**Jamie:** Those are called thumbnail views.

**Meredith:** Okay, then I want thumbnail views of all the cameras. I also want the interface for the surveillance function to have the same look and feel as all other *SafeHome* interfaces. I want it to be intuitive, meaning I don't want to have to read a manual to use it.

**Facilitator:** Good job. Now, let's go into this function in a bit more detail . . .

## Class Models

**The scene:** Ed's cubicle, as requirements modeling begins.

**The players:** Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.

**The conversation:**

[Ed has been working to extract classes from the use case template for ACS-DCV (presented in an earlier sidebar in this chapter) and is presenting the classes he has extracted to his colleagues.]

**Ed:** So when the homeowner wants to pick a camera, he or she has to pick it from a floor plan. I've defined a **FloorPlan** class. Here's the diagram.

(They look at Figure 6.10.)

**Jamie:** So **FloorPlan** is an object that is put together with walls, doors, windows, and cameras. That's what those labeled lines mean, right?

**Ed:** Yeah, they're called "associations." One class is associated with another according to the associations I've shown. [Associations are discussed in Section 6.5.5.]

**Vinod:** So the actual floor plan is made up of walls and contains cameras and sensors that are placed within those walls. How does the floor plan know where to put those objects?

**Ed:** It doesn't, but the other classes do. See the attributes under, say, **WallSegment,** which is used to build a wall. The wall segment has start and stop coordinates and the *draw()* operation does the rest.

**Jamie:** And the same goes for windows and doors. Looks like camera has a few extra attributes.

**Ed:** Yeah, I need them to provide pan and zoom info.

**Vinod:** I have a question. Why does the camera have an ID but the others don't? I notice you have an attribute called **nextWall.** How will **WallSegment** know what the next wall will be?

**Ed:** Good question, but as they say, that's a design decision, so I'm going to delay that until . . .

**Jamie:** Give me a break . . . I'll bet you've already figured it out.

**Ed (smiling sheepishly):** True, I'm gonna use a list structure which I'll model when we get to design. If you get religious about separating analysis and design, the level of detail I have right here could be suspect.

**Jamie:** Looks pretty good to me, but I have a few more questions.

(Jamie asks questions which result in minor modifications)

**Vinod:** Do you have CRC cards for each of the objects? If so, we ought to role-play through them, just to make sure nothing has been omitted.

**Ed:** I'm not quite sure how to do them.

**Vinod:** It's not hard and they really pay off. I'll show you.

## SafeHome

### CRC Models

**The scene:** Ed's cubicle, as requirements modeling begins.

**The players:** Vinod and Ed—members of the *SafeHome* software engineering team.

**The conversation:**

[Vinod has decided to show Ed how to develop CRC cards by showing him an example.]

**Vinod:** While you've been working on surveillance and Jamie has been tied up with security, I've been working on the home management function.

**Ed:** What's the status of that? Marketing kept changing its mind.

**Vinod:** Here's the first-cut use case for the whole function . . . we've refined it a bit, but it should give you an overall view . . .

**Use case:** *SafeHome* home management function.

**Narrative:** We want to use the home management interface on a PC or an Internet connection to control electronic devices that have wireless interface controllers. The system should allow me to turn specific lights on and off, to control appliances that are connected to a wireless interface, to set my heating and air conditioning system to temperatures that I define. To do this, I want to select the devices from a floor plan of the house. Each device must be identified on the floor plan. As an optional feature, I want to control all audiovisual devices—audio, television, DVD, digital recorders, and so forth.

With a single selection, I want to be able to set the entire house for various situations. One is *home*, another is *away*, a third is *overnight travel*, and a fourth is *extended travel*. All of these situations will have settings that will be applied to all devices. In the *overnight travel* and *extended travel* states, the system should turn lights on and off at random intervals (to make it look like someone is home) and control the heating and air conditioning system. I should be able to override these setting via the Internet with appropriate password protection . . .

**Ed:** The hardware guys have got all the wireless interfacing figured out?

---

**Vinod (smiling):** They're working on it; say it's no problem. Anyway, I extracted a bunch of classes for home management and we can use one as an example. Let's use the **HomeManagementInterface** class.

**Ed:** Okay . . . so the responsibilities are what . . . the attributes and operations for the class and the collaborations are the classes that the responsibilities point to.

**Vinod:** I thought you didn't understand CRC.

**Ed:** Maybe a little, but go ahead.

**Vinod:** So here's my class definition for **HomeManagementInterface.**

**Attributes:**

optionsPanel—contains info on buttons that enable user to select functionality.

situationPanel—contains info on buttons that enable user to select situation.

**Operations:**

*displayControl(), selectControl(), displaySituation(), select situation(), accessFloorplan(), selectDeviceIcon(), displayDevicePanel(), accessDevicePanel(), . . .*

**Class:** HomeManagementInterface

| **Responsibility** | **Collaborator** |
|---|---|
| *displayControl()* | **OptionsPanel** (class) |
| *selectControl()* | **OptionsPanel** (class) |
| *displaySituation()* | **SituationPanel** (class) |
| *selectSituation()* | **SituationPanel** (class) |
| *accessFloorplan()* | **FloorPlan** (class) . . . |

. . .

**Ed:** So when the operation *accessFloorplan()* is invoked, it collaborates with the **FloorPlan** object just like the one we developed for surveillance. Wait, I have a description of it here. (They look at Figure 6.10.)

## *Data Flow Modeling*

**The scene:** Jamie's cubicle, after the last requirements gathering meeting has concluded.

**The players:** Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.

**The conversation:**

(Jamie has sketched out the models shown in Figures 7.1 through 7.5 and is showing them to Ed and Vinod.)

**Jamie:** I took a software engineering course in college, and they taught us this stuff. The Prof said it's a bit old-fashioned, but you know what, it helps me to clarify things.

**Ed:** That's cool. But I don't see any classes or objects here.

**Jamie:** No . . . this is just a flow model with a little behavioral stuff thrown in.

**Vinod:** So these DFDs represent an I-P-O view of the software, right.

**Ed:** I-P-O?

**Vinod:** Input-process-output. The DFDs are actually pretty intuitive . . . if you look at 'em for a moment, they show how data objects flow through the system and get transformed as they go.

**Ed:** Looks like we could convert every bubble into an executable component . . . at least at the lowest level of the DFD.

**Jamie:** That's the cool part, you can. In fact, there's a way to translate the DFDs into an design architecture.

**Ed:** Really?

**Jamie:** Yeah, but first we've got to develop a complete requirements model and this isn't it.

**Vinod:** Well, it's a first step, but we're going to have to address class-based elements and also behavioral aspects, although the state diagram and PAT does some of that.

**Ed:** We've got a lot work to do and not much time to do it.

(Doug—the software engineering manager—walks into the cubical.)

**Doug:** So the next few days will be spent developing the requirements model, huh?

**Jamie (looking proud):** We've already begun.

**Doug:** Good, we've got a lot of work to do and not much time to do it.

(The three software engineers look at one another and smile.)

## SafeHome

### *Design versus Coding*

**The scene:** Jamie's cubicle, as the team prepares to translate requirements into design.

**The players:** Jamie, Vinod, and Ed—all members of the *SafeHome* software engineering team.

**The conversation:**

**Jamie:** You know, Doug [the team manager] is obsessed with design. I gotta be honest, what I really love doing is coding. Give me C++ or Java, and I'm happy.

**Ed:** Nah . . . you like to design.

**Jamie:** You're not listening; coding is where it's at.

**Vinod:** I think what Ed means is you don't really like coding; you like to design and express it in code. Code is the language you use to represent the design.

**Jamie:** And what's wrong with that?

**Vinod:** Level of abstraction.

**Jamie:** Huh?

**Ed:** A programming language is good for representing details like data structures and algorithms, but it's not so good for representing architecture or component-to-component collaboration . . . stuff like that.

**Vinod:** And a screwed-up architecture can ruin even the best code.

**Jamie (thinking for a minute):** So, you're saying that I can't represent architecture in code . . . that's not true.

**Vinod:** You can certainly imply architecture in code, but in most programming languages, it's pretty difficult to get a quick, big-picture read on architecture by examining the code.

**Ed:** And that's what we want before we begin coding.

**Jamie:** Okay, maybe design and coding are different, but I still like coding better.

## SafeHome

### *Design Concepts*

**The scene:** Vinod's cubicle, as design modeling begins.

**The players:** Vinod, Jamie, and Ed—members of the *SafeHome* software engineering team. Also, Shakira, a new member of the team.

**The conversation:**

[All four team members have just returned from a morning seminar entitiled "Applying Basic Design Concepts," offered by a local computer science professor.]

**Vinod:** Did you get anything out of the seminar?

**Ed:** Knew most of the stuff, but it's not a bad idea to hear it again, I suppose.

**Jamie:** When I was an undergrad CS major, I never really understood why information hiding was as important as they say it is.

**Vinod:** Because . . . bottom line . . . it's a technique for reducing error propagation in a program. Actually, functional independence also accomplishes the same thing.

**Shakira:** That's used in Extreme Programming, I think she said.

**Ed:** Yep. It's not a whole lot different than refinement, only you do it after the design or code is completed. Kind of an optimization pass through the software, if you ask me.

**Shakira:** I wasn't a CS grad, so a lot of the stuff the instructor mentioned is new to me. I can generate good code and fast. I don't see why this stuff is so important.

**Jamie:** I've seen your work, Shak, and you know what, you do a lot of this stuff naturally . . . that's why your designs and code work.

**Shakira (smiling):** Well, I always do try to partition the code, keep it focused on one thing, keep interfaces simple and constrained, reuse code whenever I can . . . that sort of thing.

**Ed:** Modularity, functional independence, hiding, patterns . . . see.

**Jamie:** I still remember the very first programming course I took . . . they taught us to refine the code iteratively.

**Vinod:** Same thing can be applied to design, you know.

**Vinod:** The only concepts I hadn't heard of before were "aspects" and "refactoring."

**Jamie:** Let's get back to *SafeHome* design. I think we should put these concepts on our review checklist as we develop the design model for *SafeHome*.

**Vinod:** I agree. But as important, let's all commit to think about them as we develop the design.

## SafeHome

### *Refining an Analysis Class into a Design Class*

**The scene:** Ed's cubicle, as design modeling begins.

**The players:** Vinod and Ed—members of the *SafeHome* software engineering team.

**The conversation:**

[Ed is working on the **FloorPlan** class (see sidebar discussion in Section 6.5.3 and Figure 6.10) and has refined it for the design model.]

**Ed:** So you remember the **FloorPlan** class, right? It's used as part of the surveillance and home management functions.

**Vinod (nodding):** Yeah, I seem to recall that we used it as part of our CRC discussions for home management.

**Ed:** We did. Anyway, I'm refining it for design. Want to show how we'll actually implement the **FloorPlan** class. My idea is to implement it as a set of linked lists [a specific data structure] So . . . I had to refine the analysis class **FloorPlan** (Figure 6.10) and actually, sort of simplify it.

**Vinod:** The analysis class showed only things in the problem domain, well, actually on the computer screen, that were visible to the end user, right?

**Ed:** Yep, but for the **FloorPlan** design class, I've got to add some things that are implementation specific. I needed to show that **FloorPlan** is an aggregation of segments—hence the **Segment** class—and that the **Segment** class is composed of lists for wall segments, windows, doors, and so on. The class **Camera** collaborates with **FloorPlan,** and obviously, there can be many cameras in the floor plan.

**Vinod:** Phew, let's see a picture of this new **FloorPlan** design class.

[Ed shows Vinod the drawing shown in Figure 8.3.]

**Vinod:** Okay, I see what you're trying to do. This allows you to modify the floor plan easily because new items can be added to or deleted from the list—the aggregation—without any problems.

**Ed (nodding):** Yeah, I think it'll work.

**Vinod:** So do I.

## SafeHome

### *Choosing an Architectural Style*

**The scene:** Jamie's cubicle, as design modeling begins.

**The players:** Jamie and Ed—members of the *SafeHome* software engineering team.

**The conversation:**

**Ed (frowning):** We've been modeling the security function using UML . . . you know classes, relationships, that sort of stuff. So I guess the object-oriented architecture[3] is the right way to go.

**Jamie:** But . . .?

**Ed:** But . . . I have trouble visualizing what an object-oriented architecture is. I get the call and return architecture, sort of a conventional process hierarchy, but OO . . . I don't know, it seems sort of amorphous.

**Jamie (smiling):** Amorphous, huh?

**Ed:** Yeah . . . what I mean is I can't visualize a real structure, just design classes floating in space.

**Jamie:** Well, that's not true. There are class hierarchies . . . think of the hierarchy (aggregation) we did for the **FloorPlan** object [Figure 8.3]. An OO architecture is a combination of that structure and the interconnections—you know, collaborations—between the classes. We can show it by fully describing the attributes and operations, the messaging that goes on, and the structure of the classes.

**Ed:** I'm going to spend an hour mapping out a call and return architecture; then I'll go back and consider an OO architecture.

**Jamie:** Doug'll have no problem with that. He said that we should consider architectural alternatives. By the way, there's absolutely no reason why both of these architectures couldn't be used in combination with one another.

**Ed:** Good. I'm on it.

## SafeHome

### *Architecture Assessment*

**The scene:** Doug Miller's office as architectural design modeling proceeds.

**The players:** Vinod, Jamie, and Ed—members of the *SafeHome* software engineering team and Doug Miller, manager of the software engineering group.

**The conversation:**

**Doug:** I know you guys are deriving a couple of different architectures for the *SafeHome* product, and that's a good thing. I guess my question is, how are we going to choose the one that's best?

**Ed:** I'm working on a call and return style and then either Jamie or I are going to derive an OO architecture.

**Doug:** Okay, and how do we choose?

**Jamie:** I took a CS course in design in my senior year, and I remember that there are a number of ways to do it.

**Vinod:** There are, but they're a bit academic. Look, I think we can do our assessment and choose the right one using use cases and scenarios.

**Doug:** Isn't that the same thing?

**Vinod:** Not when you're talking about architectural assessment. We already have a complete set of use cases. So we apply each to both architectures and see how the system reacts, how components and connectors work in the use case context.

**Ed:** That's a good idea. Makes sure we didn't leave anything out.

**Vinod:** True, but it also tells us whether the architectural design is convoluted, whether the system has to twist itself into a pretzel to get the job done.

**Jamie:** Scenarios aren't just another name for use cases.

**Vinod:** No, in this case a scenario implies something different.

**Doug:** You're talking about a quality scenario or a change scenario, right?

**Vinod:** Yes. What we do is go back to the stakeholders and ask them how *SafeHome* is likely to change over the next, say, three years. You know, new versions, features, that sort of thing. We build a set of change scenarios. We also develop a set of quality scenarios that define the attributes we'd like to see in the software architecture.

**Jamie:** And we apply them to the alternatives.

**Vinod:** Exactly. The style that handles the use cases and scenarios best is the one we choose.

## SafeHome

### *Refining a First-Cut Architecture*

**The scene:** Jamie's cubicle, as design modeling begins.

**The players:** Jamie and Ed—members of the *SafeHome* software engineering team.

**The conversation:**

[Ed has just completed a first-cut design of the monitor sensors subsystem. He stops in to ask Jamie her opinion.]

**Ed:** So here's the architecture that I derived.

[Ed shows Jamie Figure 9.16, which she studies for a few moments.]

**Jamie:** That's cool, but I think we can do a few things to make it simpler . . . and better.

**Ed:** Such as?

**Jamie:** Well, why did you use the *sensor input controller* component?

**Ed:** Because you need a controller for the mapping.

**Jamie:** Not really. The controller doesn't do much, since we're managing a single flow path for incoming data. We can eliminate the controller with no ill effects.

**Ed:** I can live with that. I'll make the change and . . .

**Jamie (smiling):** Hold up! We can also implode the components *establish alarm conditions* and *select phone number*. The transform controller you show isn't really necessary, and the small decrease in cohesion is tolerable.

**Ed:** Simplification, huh?

**Jamie:** Yep. And while we're making refinements, it would be a good idea to implode the components *format display* and *generate display*. Display formatting for the control panel is simple. We can define a new module called *produce display*.

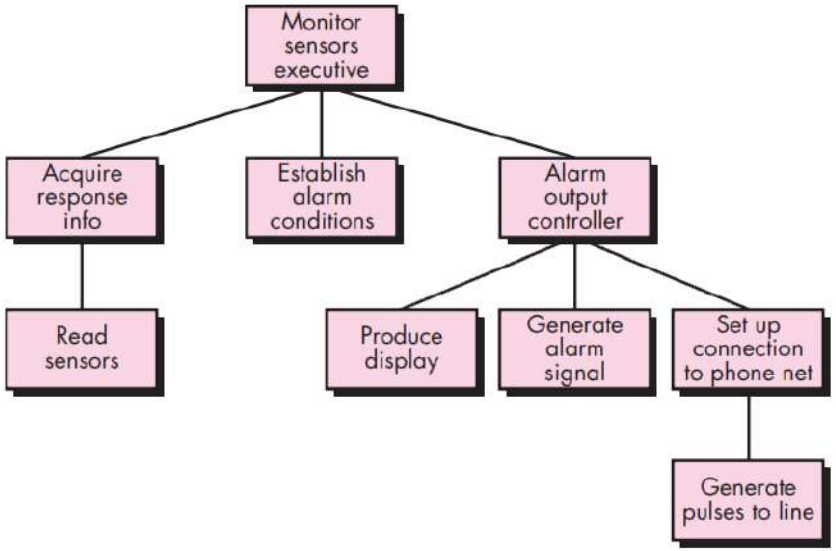
**Ed (sketching):** So this is what you think we should do?"

[Shows Jamie Figure 9.17.]

**Jamie:** It's a good start.

**FIGURE 9.17**

**Refined program structure for *monitor sensors***



## SafeHome

### *The OCP in Action*

**The scene:** Vinod's cubicle.

**The players:** Vinod and Shakira—members of the *SafeHome* software engineering team.

**The conversation:**

**Vinod:** I just got a call from Doug [the team manager]. He says marketing wants to add a new sensor.

**Shakira (smirking):** Not again, jeez!

**Vinod:** Yeah . . . and you're not going to believe what these guys have come up with.

**Shakira:** Amaze me.

**Vinod (laughing):** They call it a doggie angst sensor.

**Shakira:** Say what?

**Vinod:** It's for people who leave their pets home in apartments or condos or houses that are close to one another. The dog starts to bark. The neighbor gets angry and complains. With this sensor, if the dog barks for more than, say, a minute, the sensor sets a special alarm mode that calls the owner on his or her cell phone.

**Shakira:** You're kidding me, right?

**Vinod:** Nope. Doug wants to know how much time it's going to take to add it to the security function.

**Shakira (thinking a moment):** Not much . . . look. [She shows Vinod Figure 10.4] We've isolated the actual sensor classes behind the **sensor** interface. As long as we have specs for the doggie sensor, adding it should be a piece of cake. Only thing I'll have to do is create an appropriate component . . . uh, class, for it. No change to the **Detector** component at all.

**Vinod:** So I'll tell Doug it's no big deal.

**Shakira:** Knowing Doug, he'll keep us focused and not deliver the doggie thing until the next release.

**Vinod:** That's not a bad thing, but you can implement now if he wants you to?

**Shakira:** Yeah, the way we designed the interface lets me do it with no hassle.

**Vinod (thinking a moment):** Have you ever heard of the open-closed principle?

**Shakira (shrugging):** Never heard of it.

**Vinod (smiling):** Not a problem.

## SafeHome

### *Cohesion in Action*

**The scene:** Jamie's cubicle.

**The players:** Jamie and Ed—members of the *SafeHome* software engineering team who are working on the surveillance function.

**The conversation:**

**Ed:** I have a first-cut design of the **camera** component.

**Jamie:** Wanna do a quick review?

**Ed:** I guess . . . but really, I'd like your input on something.

(Jamie gestures for him to continue.)

**Ed:** We originally defined five operations for **camera**. Look . . .

*determineType()* tells me the type of camera.

*translateLocation()* allows me to move the camera around the floor plan.

*displayID()* gets the camera ID and displays it near the camera icon.

*displayView()* shows me the field of view of the camera graphically.

*displayZoom()* shows me the magnification of the camera graphically.

**Ed:** I've designed each separately, and they're pretty simple operations. So I thought it might be a good idea to combine all of the display operations into just one that's called *displayCamera()*—it'll show the ID, the view, and the zoom. Whaddaya think?

**Jamie (grimacing):** Not sure that's such a good idea.

**Ed (frowning):** Why, all of these little ops can cause headaches.

**Jamie:** The problem with combining them is we lose cohesion, you know, the *displayCamera()* op won't be single-minded.

**Ed (mildly exasperated):** So what? The whole thing will be less than 100 source lines, max. It'll be easier to implement, I think.

**Jamie:** And what if marketing decides to change the way that we represent the view field?

**Ed:** I just jump into the *displayCamera()* op and make the mod.

**Jamie:** What about side effects?

**Ed:** Whaddaya mean?

**Jamie:** Well, say you make the change but inadvertently create a problem with the ID display.

**Ed:** I wouldn't be that sloppy.

**Jamie:** Maybe not, but what if some support person two years from now has to make the mod. He might not understand the op as well as you do, and, who knows, he might be sloppy.

**Ed:** So you're against it?

**Jamie:** You're the designer . . . it's your decision . . . just be sure you understand the consequences of low cohesion.

**Ed (thinking a moment):** Maybe we'll go with separate display ops.

**Jamie:** Good decision.

## SafeHome

### *Coupling in Action*

**The scene:** Shakira's cubicle.

**The players:** Vinod and Shakira—members of the *SafeHome* software team who are working on the security function.

**The conversation:**

**Shakira:** I had what I thought was a great idea . . . then I thought about it a little, and it seemed like a not so great idea. I finally rejected it, but I just thought I'd run it by you.

**Vinod:** Sure. What's the idea?

**Shakira:** Well, each of the sensors recognizes an alarm condition of some kind, right?

**Vinod (smiling):** That's why we call them sensors, Shakira.

**Shakira (exasperated):** Sarcasm, Vinod, you've got to work on your interpersonal skills.

**Vinod:** You were saying?

**Shakira:** Okay, anyway, I figured . . . why not create an operation within each sensor object called *makeCall()* that would collaborate directly with the **OutgoingCall** component, well, with an interface to the **OutgoingCall** component.

**Vinod (pensive):** You mean rather than having that collaboration occur out of a component like **ControlPanel** or something?

**Shakira:** Yeah . . . but then I said to myself, that means that every sensor object will be connected to the **OutgoingCall** component, and that means that its indirectly coupled to the outside world and . . . well, I just thought it made things complicated.

**Vinod:** I agree. In this case, it's a better idea to let the sensor interface pass info to the **ControlPanel** and let it initiate the outgoing call. Besides, different sensors might result in different phone numbers. You don't want the senor to store that information because if it changes . . .

**Shakira:** It just didn't feel right.

**Vinod:** Design heuristics for coupling tell us it's not right.

**Shakira:** Whatever . . .

## SafeHome

### *Violating a UI Golden Rule*

**The scene:** Vinod's cubicle, as user interface design begins.

**The players:** Vinod and Jamie, members of the *SafeHome* software engineering team.

**The conversation:**

**Jamie:** I've been thinking about the surveillance function interface.

**Vinod (smiling):** Thinking is good.

**Jamie:** I think maybe we can simplify matters some.

**Vinod:** Meaning?

**Jamie:** Well, what if we eliminate the floor plan entirely. It's flashy, but it's going to take serious development effort. Instead we just ask the user to specify the camera he wants to see and then display the video in a video window.

**Vinod:** How does the homeowner remember how many cameras are set up and where they are?

**Jamie (mildly irritated):** He's the homeowner; he should know.

**Vinod:** But what if he doesn't?

**Jamie:** He should.

---

**Vinod:** That's not the point . . . what if he forgets?

**Jamie:** Uh, we could provide a list of operational cameras and their locations.

**Vinod:** That's possible, but why should he have to ask for a list?

**Jamie:** Okay, we provide the list whether he asks or not.

**Vinod:** Better. At least he doesn't have to remember stuff that we can give him.

**Jamie (thinking for a moment):** But you like the floor plan, don't you?

**Vinod:** Uh huh.

**Jamie:** Which one will marketing like, do you think?

**Vinod:** You're kidding, right?

**Jamie:** No.

**Vinod:** Duh . . . the one with the flash . . . they love sexy product features . . . they're not interested in which is easier to build.

**Jamie (sighing):** Okay, maybe I'll prototype both.

**Vinod:** Good idea . . . then we let the customer decide.

## *Use Cases for UI Design*

**The scene:** Vinod's cubicle, as user interface design continues.

**The players:** Vinod and Jamie, members of the *SafeHome* software engineering team.

**The conversation:**

**Jamie:** I pinned down our marketing contact and had her write a use case for the surveillance interface.

**Vinod:** From whose point of view?

**Jamie:** The homeowner, who else is there?

**Vinod:** There's also the system administrator role, even if it's the homeowner playing the role, it's a different point of view. The "administrator" sets the system up, configures stuff, lays out the floor plan, places the cameras . . .

**Jamie:** All I had her do was play the role of the homeowner when he wants to see video.

**Vinod:** That's okay. It's one of the major behaviors of the surveillance function interface. But we're going to have to examine the system administration behavior as well.

**Jamie (irritated):** You're right.

[Jamie leaves to find the marketing person. She returns a few hours later.]

**Jamie:** I was lucky, I found her and we worked through the administrator use case together. Basically, we're going to define "administration" as one function that's applicable to all other *SafeHome* functions. Here's what we came up with.

[Jamie shows the informal use case to Vinod.]

**Informal use case:** I want to be able to set or edit the system layout at any time. When I set up the system, I select an administration function. It asks me whether I want to do a new setup or whether I want to edit an existing setup. If I select a new setup, the system displays a drawing screen that will enable me to draw the floor plan onto a grid. There will be icons for walls, windows, and doors so that drawing is easy. I just stretch the icons to their appropriate lengths. The system will display the lengths in feet or meters (I can select the measurement system). I can select from a library of sensors and cameras and place them on the floor plan. I get to label each, or the system will do automatic labeling. I can establish settings for sensors and cameras from appropriate menus. If I select edit, I can move sensors or cameras, add new ones or delete existing ones, edit the floor plan, and edit the settings for cameras and sensors. In every case, I expect the system to do consistency checking and to help me avoid mistakes.

**Vinod (after reading the scenario):** Okay, there are probably some useful design patterns [Chapter 12] or reusable components for GUIs for drawing programs. I'll betcha 50 bucks we can implement some or most of the administrator interface using them.

**Jamie:** Agreed. I'll check it out.

## *Interface Design Review*

**The scene:** Doug Miller's office.

**The players:** Doug Miller (manager of the *SafeHome* software engineering group) and Vinod Raman, a member of the *SafeHome* product software engineering team.

**The conversation:**

**Doug:** Vinod, have you and the team had a chance to review the **SafeHomeAssured.com** e-commerce interface prototype?

**Vinod:** Yeah . . . we all went through it from a technical point of view, and I have a bunch of notes. I e-mailed 'em to Sharon [manager of the WebApp team for the outsourcing vendor for the *SafeHome* e-commerce website] yesterday.

**Doug:** You and Sharon can get together and discuss the small stuff . . . give me a summary of the important issues.

**Vinod:** Overall, they've done a good job, nothing ground breaking, but it's a typical e-commerce interface, decent aesthetics, reasonable layout, they've hit all the important functions . . .

**Doug (smiling ruefully):** But?

**Vinod:** Well, there are a few things . . . .

**Doug:** Such as . . .

**Vinod (showing Doug a sequence of story-boards for the interface prototype):** Here's the major functions menu that's displayed on the home page:

**Learn about *SafeHome*.**

**Describe your home.**

**Get *SafeHome* component recommendations.**

**Purchase a *SafeHome* system.**

**Get technical support.**

The problem isn't with these functions. They're all okay, but the level of abstraction isn't right.

**Doug:** They're all major functions, aren't they?

**Vinod:** They are, but here's the thing . . . you can purchase a system by inputting a list of components . . . no real need to describe the house if you don't want to. I'd suggest only four menu options on the home page:

**Learn about *SafeHome*.**

**Specify the *SafeHome* system you need.**

**Purchase a *SafeHome* system.**

**Get technical support.**

When you select **Specify the *SafeHome* system you need**, you'll then have the following options:

**Select *SafeHome* components.**

**Get *SafeHome* component recommendations.**

If you're a knowledgeable user, you'll select components from a set of categorized pull-down menus for sensors, cameras, control panels, etc. If you need help, you'll ask for a recommendation and that will require that you describe your house. I think it's a bit more logical.

**Doug:** I agree. Have you talked with Sharon about this?

**Vinod:** No, I want to discuss this with marketing first; then I'll give her a call.

## SafeHome

### *Applying Patterns*

**The scene:** Informal discussion during the design of a software increment that implements sensor control via the Internet for **SafeHomeAssured.com.**

**The players:** Jamie (responsible for design) and Vinod (**SafeHomeAssured.com** chief system architect).

**The conversation:**

**Vinod:** So how is the design of the camera control interface coming along?

**Jamie:** Not too bad. I've designed most of the capability to connect to the actual sensors without too many problems. I've also started thinking about the interface for the users to actually move, pan, and zoom the cameras from a remote Web page, but I'm not sure I've got it right yet.

**Vinod:** What have you come up with?

**Jamie:** Well, the requirements are that the camera control needs to be highly interactive—as the user moves the control, the camera should move as soon as possible. So, I was thinking of having a set of buttons laid out like a normal camera, but when the user clicks them, it controls the camera.

**Vinod:** Hmmm. Yeah, that would work, but I'm not sure it's right—for each click of a control you need to wait for the whole client-server communication to occur, and so you won't get a good sense of quick feedback.

**Jamie:** That's what I thought—and why I wasn't very happy with the approach, but I'm not sure how else I might do it.

**Vinod:** Well, why not just use the **InteractiveDeviceControl** pattern!

**Jamie:** Uhmmm—what's that? I haven't heard of it?

**Vinod:** It's basically a pattern for exactly the problem you are describing. The solution it proposes is basically to create a control connection to the server with the device, through which control commands can be sent. That way you don't need to send normal HTTP requests. And the pattern even shows how you can implement this using some simple AJAX techniques. You have some simple client-side JavaScript that communicates directly with the server and sends the commands as soon as the user does anything.

**Jamie:** Cool! That's just what I needed to solve this thing. Where do I find it?

**Vinod:** It's available in an online repository. Here's the URL.

**Jamie:** I'll go check it out.

**Vinod:** Yep—but remember to check the consequences field for the pattern. I seem to remember that there was something in there about needing to be careful about issues of security. I think it might be because you are creating a separate control channel and so bypassing the normal Web security mechanisms.

**Jamie:** Good point. I probably wouldn't have thought of that! Thanks.

## Quality Issues

**The scene:** Doug Miller's office as the *SafeHome* software project begins.

**The players:** Doug Miller (manager of the *SafeHome* software engineering team) and other members of the product software engineering team.

**The conversation:**

**Doug:** I know we didn't spend time developing a quality plan for this project, but we're already into it and we have to consider quality … right?

**Jamie:** Sure. We've already decided that as we develop the requirements model [Chapters 6 and 7], Ed has committed to develop a testing procedure for each requirement.

**Doug:** That's really good, but we're not going to wait until testing to evaluate quality, are we?

**Vinod:** No! Of course not. We've got reviews scheduled into the project plan for this software increment. We'll begin quality control with the reviews.

**Jamie:** I'm a bit concerned that we won't have enough time to conduct all the reviews. In fact, I know we won't.

**Doug:** Hmmm. So what do you propose?

**Jamie:** I say we select those elements of the requirements and design model that are most critical to *SafeHome* and review them.

**Vinod:** But what if we miss something in a part of the model we don't review?

**Shakira:** I read something about a sampling technique [Section 15.6.4] that might help us target candidates for review. (Shakira explains the approach.)

**Jamie:** Maybe … but I'm not sure we even have time to sample every element of the models.

**Vinod:** What do you want us to do, Doug?

**Doug:** Let's steal something from Extreme Programming [Chapter 3]. We'll develop the elements of each model in pairs—two people—and conduct an informal review of each as we go. We'll then target "critical" elements for a more formal team review, but keep those reviews to a minimum. That way, everything gets looked at by more than one set of eyes, but we still maintain our delivery dates.

**Jamie:** That means we're going to have to revise the schedule.

**Doug:** So be it. Quality trumps schedule on this project.