Scenarios and Use Cases

Daniel M. Berry

© 2004 Daniel M. Berry

Using a Computer-Based System

Consider an interactive computer-based system (CBS), *S*. It is called interactive because its users interact with it.

A user tends to think of *S* in the ways that he or she will use it.

Using a CBS -1

For example, if *S* is the *Sensus* voting system, the average voter user tends to think of two particular ways to use *S*:

- **registering to vote, and**
- **voting, i.e., marking a ballot in an election.**

Using a CBS -2

There are other users of this *S* that think of other particular ways to use the *S*.

For example, the voting manager user has four other particular ways to use *S*:

- **preparing a ballot for a particular election,**
- **opening the polls for a particular election for voting by voters,**
- **closing the polls for a particular election,**
- **counting the votes for a particular election.**

Using a CBS -3

Of course, the voting manager is also a voter and can also register to vote and vote.

Use Case

Each such particular way to use *S* is called a *use case*.

It is one case of the many ways to use *S*.

Scenario

A use case should not be confused with a closely related concept called a *scenario*. A scenario of *S* is a particular sequence of interaction steps between a user of *S* and *S*.

Use Cases and Scenarios

The relation between use cases and scenarios can make both terms clearer.

A single use case contains many, many scenarios.

A use case *UC* of *S* has a so-called *typical* scenario. This scenario is that identified by the stakeholders of *S* as being the normal case of *UC* that proceeds with all decisions being made in the so-called normal or typical way.

Variations

***UC* consists also of variations called *alternatives* and *exceptions*.**

Alternatives -1

An alternative of *UC* is a sub-use-case, which may include many scenarios, that achieves the main goal of *UC* through different sequences of steps or fails to achieve the goals of *UC* although it follows most of the steps of *UC*.

Alternatives -2

For example, for the *voting* use case, the typical case is voting in one session.

One alternative is voting in multiple sessions.

Another alternative is starting to vote, but not finishing.

Exceptions

An exception of *UC* is a sub-use-case, which may include many scenarios, that deals with the conditions along the typical scenario and other sub-use-cases that differ from the norm and those already covered.

For example, for the *voting*, what to do with a non-registered voter trying to vote is an exception.

Misuse Cases

(Sindre and Opdahl) and Alexander have proposed *misuse* cases to capture use cases that a system must be protected against, that you do not want to happen at all, e.g., a security breach.

Sharing of Subsequences

Observe that all the scenarios of a use case share many subsequences of steps.

Some sets of these subsequences of steps may constitute sub-use-cases that are worth considering as use cases that can be used by other use cases.

For example, the *voting* and *registering* use cases may share the steps for validating that a user is a registered voter.

To Precise Definitions

After this intuitive description of use cases and scenarios, it is possible to give more precise definitions.

While the intuitive discussion considers scenarios as elements of a use case, the precise definitions consider use cases as generalizations of scenarios.

Scenario — Definition

A scenario of system *S* is a detailed description, often in natural language prose, of the sequence of steps constituting one use of *S* by a user for achievement of a specific objective.

Use Case — Definition

A use case of *S* is a parameterized abstraction of one or more scenarios such that each scenario is obtained by instantiating the use case with one particular collection of parameter values.

Some collections of parameter values may yield scenarios that were not in the original set being abstracted by the use case.

Purpose of Ss & UCs

Thus, scenarios and use cases are means to document the way users use *S* in order to do their work.

Scenarios and use cases of *S* help to identify requirements of *S* in the sense that *S* must be able to do the system's part of every scenario and instantiated use case.

Elicitation of Ss & UCs

Scenarios and use cases are considered good vehicles for getting users to describe how they use or will use systems.

Observation of Ss & UCs

Scenarios and use cases can be inferred by requirements engineers from their in situ observations of the work place.

Other Uses of Ss & UCs

Scenarios and use cases of *S* form a good basis for writing user's manuals and on-line help about *S* since all must describe how the users use *S*.

Scenarios and use cases of *S* form a good basis for building covering test cases for testing *S* in the sense that all must cover all the ways the users will use *S*.

To Examples

Now let us identify the use cases and scenarios of the *Sensus* system whose classes we identified before.

Recall...

Sensus Modules

Sensus has four main modules:

- ***Registrar — The registrar registers voters prior to an election.***
- ***Pollster — The pollster acts as a voters' [sic] agent, presenting human readable ballots to a voter, collecting the voter's responses to ballot questions, performing cryptographic functions on the voter's behalf, obtaining necessary validations and receipts, and delivering ballots to the ballot box....***

The pollster is the only component of the Sensus system that voters must trust completely; voters concerned about the privacy of their ballots may want to install personal copies of the pollster on trusted machines.

- ***Validator — The validator ensures that only registered voters can vote, and that only one ballot is counted for each registered voter.***
- ***Tallier — The tallier tallies the results of the election or survey.*** **[The word "tallier" should be read as "tally-er".]**

Registering to Vote

Before registering to vote, a voter must obtain a voter identification number, token, and registration address from the election administrators.

You may begin the registration process by running the pollster module. This is generally done by invoking the* sensus *command.

The pollster module will display a menu of options. Select the "register to vote" option.

The pollster will generate a public/private key pair for you and then prompt you for your identification number, token, and the registration address.

The pollster will prepare a registration request on your behalf and submit it to the registrar. If all goes well, the pollster will collect an acknowledgment from the registrar within a few seconds. Then, the pollster will prompt you for a file name for saving your registration information. Select a name you will remember, as you will need to tell the pollster

the name of your registration file every time you vote. If you are registered with more than one election authority, make sure you store your registration information in separate files. All Sensus files will be stored in your **.sensus** *directory; if you do not have one, the pollster will create one for you.*

Marking Your Ballot

Before you can mark a ballot, you must obtain the unvoted ballot for the election and place it in your* .sensus *directory. You must also be registered to vote in that election.

Start by running the pollster module as you did when you registered to vote.

If you would like to review the ballot before you mark it, select "view ballot questions and instructions" from the pollster menu.

When you are ready to mark your ballot, select "mark ballot" from the pollster menu.

The pollster will prompt you for the name of the ballot and your registration file name.

The pollster will then display the ballot questions one at a time along with instructions for responding to each question.

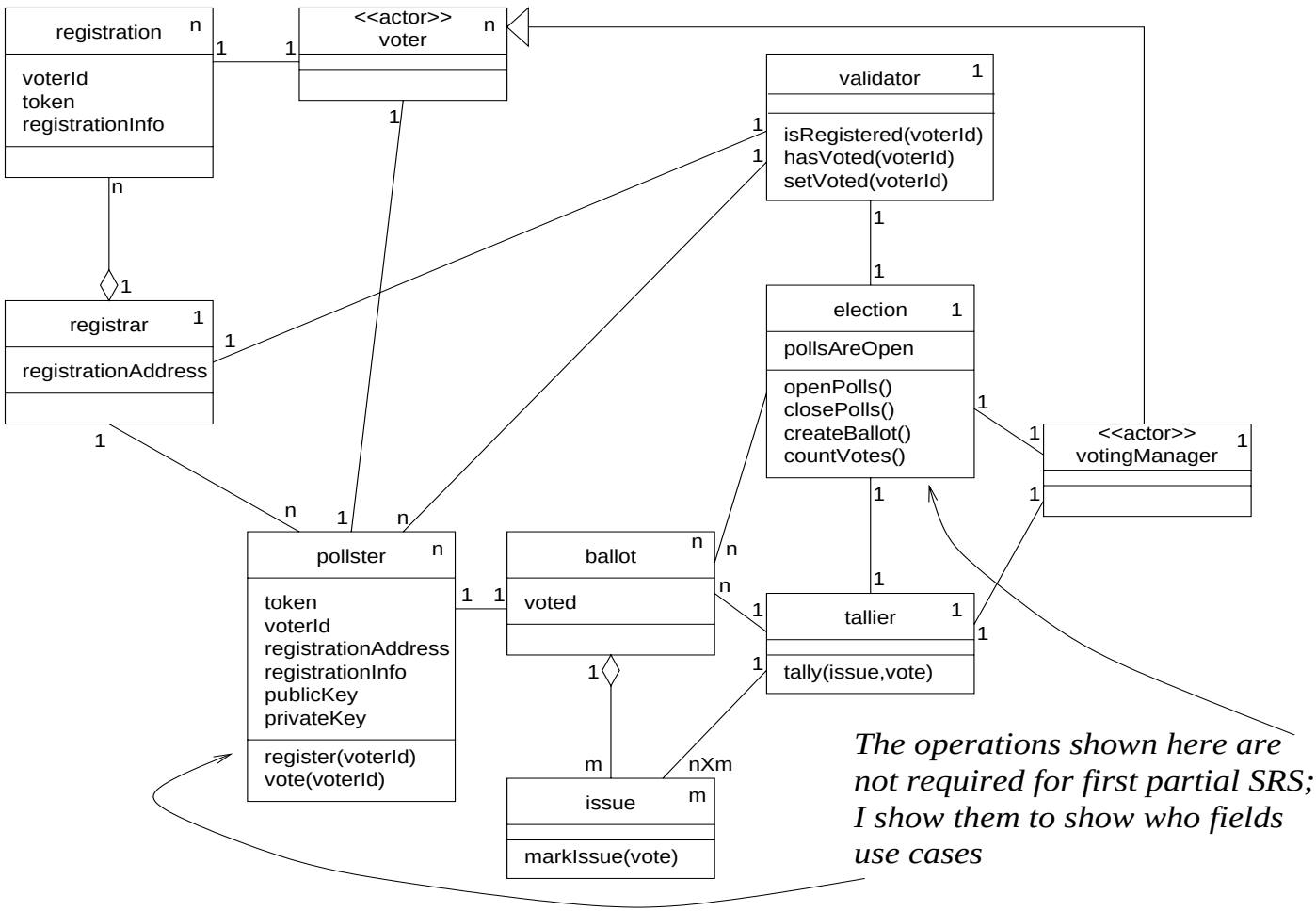
[Why not display before, when viewing?]

If you change your mind or make a mistake marking your ballot, you can remark your ballot. At this time it is not possible to change your response to some ballot questions without remarking your entire ballot.

When you have finished marking your ballot, the pollster will prompt you to continue the voting process. By answering yes at each of the prompts, you can authorize the pollster to complete the entire voting process on your behalf immediately. This process usually takes a few minutes. If you do not want to

complete the process right away, you can exit from the pollster program and run it again later to pick up where you left off.

Class Diagram



The operations shown here are not required for first partial SRS; I show them to show who fields use cases

Use Cases

The author of the *Sensus* description has provided us for free two use cases, namely for

- **registering to vote**
- **marking a ballot (or voting)**

Natural Language Description

Since we are working with clients and users, the preferred initial representation of use cases is in natural language.

We use numbered natural language sentences arranged in multiple columns.

Multiple Columns

Each column contains the steps done by one class instance in the system or environment.

A class instance in the environment is called an actor and a class instance in the system is called a process.

UC “Registering to Vote”

Voter	Pollster	Registrar
1. Before registering to vote, a voter must have obtained a voter [identification number, token, and registration address] from the election administrators.		
2. Voter invokes the *sensus* command, to run the Pollster.		
	3. Pollster displays a menu of options.	
4. Voter selects the ‘‘register to vote’’ option.		
	5. Pollster generates and sends a public/private key pair for Voter.	
	6. Pollster prompts voter for his/her [identification number, token, and registration address].	

Voter	Pollster	Registrar
7. Voter sends his/her [identification number, token, and registration address].		
	8. Pollster prepares registration request on Voter's behalf.	
	9. Pollster submits registration request to registrar	
		10. If all goes well, Registrar sends acknowledgement to Pollster within a few seconds.
	11. Pollster prompts Voter for a file name for saving his/her registration information.	
12. Voter selects a name he/she will remember; this name must be unique for the registration authority.		

Voter	Pollster	Registrar
13. Voter sends selected name to Pollster.		
	14. If Voter has a *.sensus* directory, Pollster creates file with selected name in Voter's *.sensus* directory and stores registration information in the file.	
15. Voter exits *sensus* program.		

EXCEPTION for step 10:

Voter	Pollster	Registrar
		10. If all does not go well, ...

EXCEPTION for step 14:

Voter	Pollster	Registrar
	14. If Voter does not have a *.sensus* directory, Pollster creates one and then Pollster creates file with selected name in Voter's *.sensus* directory and stores registration information in the file.	

ALTERNATIVES for step 7:

Voter	Pollster	Registrar
7. Voter sends his/her [token, identification number, and registration address].		

... one for each permutation of the data...

UC “Marking your Ballot”

Voter	Pollster	Election	Registrar
1. Voter invokes the *sensus* command, to run the Pollster.			
	2. Pollster displays a menu of options		
3. Voter slects the ‘‘marking ballot’’ option.			
	4. Pollster generates and sends a public/private key pair for Voter.		
	5. Pollster prompts voter for his/her [identification number, token, and registration address].		

Voter	Pollster	Election	Registrar
6. Voter sends his/her [identification number, token, and registration address].			
	7. Pollster requests verification of voterId from Registrar.		
			8. If voterId is registered, Registrar sends Pollster acknowledgement.
	9. Pollster requests an unvoted ballot from Election authority to send it to Voter.		
		10. Election authority sends an unvoted ballot to Pollster	
	11. Pollster sends unvoted ballot to Voter.		

Voter	Pollster	Election	Registrar
	12. Pollster displays pollster menu.		
13. Voter selects ‘‘view ballot questions and instructions’’ from the pollster menu.			
	14. Pollster displays ballot questions and instructions.		
15. Voter selects ‘‘mark ballot’’ from the pollster menu.			
	16. Pollster prompts for the name of the ballot and Voter’s registration file name.		
17. Voter sends the name of the ballot and Voter’s registration file name.			

Voter	Pollster	Election	Registrar
	18. Pollster displays ballot question 1 and its voting instructions.		
19. Voter votes on ballot question 1.			
...			
	20. Pollster displays ballot question *m* and its voting instructions.		
21. Voter votes on ballot question *m*.			
	22. Pollster prompts Voter with vote-ending prompt 1.		
23. Voter answers “yes” to prompt 1.			
...			

Voter	Pollster	Election	Registrar
	24. Pollster prompts Voter with vote-ending prompt *k*.		
25. Voter answers “yes” to prompt *k*.			
	26. Pollster sends completely voted ballot to Election Authority.		
27. Voter exits *sensus* program.			

ALTERNATIVE for steps 13–14 for Voter that does not want to review the ballot before marking it:

Voter	Pollster	Election	Registrar
After step 12 comes step 15.			

ALTERNATIVE for steps 18–21 for Voter that quits after some number of ballot questions and goes back to remark the entire ballot, because he or she has changed his or her mind or has made a mistake:

Voter	Pollster	Election	Registrar
After last step among 18–21, go to step 15.			

EXCEPTIONS for steps 22–25:

Voter	Pollster	Election	Registrar
Voter answers “no” to a vote-ending prompt *i*<*k* and exits the *sensus* program, able to continue from the last completed step.			

EXCEPTION for step 8:

Voter	Pollster	Election	Registrar
voterId is not registered ...			

ALTERNATIVE for any Voter step:

Voter	Pollster	Election	Registrar
Voter exits *sensus* program, able to continue from the last completed step.			

Gold Mine -1

These UCs & Ss are a gold mine of clarifying questions that an observant, thinking requirements engineer can ask of the clients and users.

Gold Mine -2

Notice how my scenarios are more complete than those produced by the client.

The client is talking off the top of his or her head.

I have time to think and to notice omissions, gaps, overlookings, etc.

I fill them in in an manner that is consistent with my understanding.

Gold Mine -3

Then, I ask the client if this is what he or she meant.

Thus, I validate my understanding and my partial model.

The client's answers help me adjust my understanding and model.

Aliases

I noticed some different terms for what might be the same thing, e.g.

- ***sensus* main menu,**
- **pollster's menu.**

I leave them separate for now, but ask the client:

- **Are they the same?**
- **Are they different?**

Again, I adjust my understanding based on the client's answers.

Ignorance

Client used terms I do not understand, e.g., public/private key pair.

He or she is asked to explain.

Are they sent to voter?

Does voter have to remember and use them each time?

Are they generated for each *sensus* session?

Inconsistency

Client was inconsistent?

Why prompt voter to continue voting process when voting of issues is done?

Especially, when right after that prompt, a series of authorizing questions is asked. These are not continued voting????

Maybe word should be “finish” or “conclude” or “authorize”?

Ask, learn, and adjust understanding.

Omissions

Client left out details?

What are these authorizing questions?

Ask, learn, and adjust understanding.

Actors and Objects

In these scenarios, you see also the actors and system objects, and thus, the classes of the class diagram you need to build.

Value of Questions -1

Inconsistencies,

incompletenesses,

aliases,

etc.

all prompt clarifying questions!

Value of Questions -2

Do you see the gold mine?

So, you should sit down with your problem description and begin to extract UCs & Ss.

Note all the problems, questions, inconsistencies, incompletenesses, etc.

Value of Questions -2

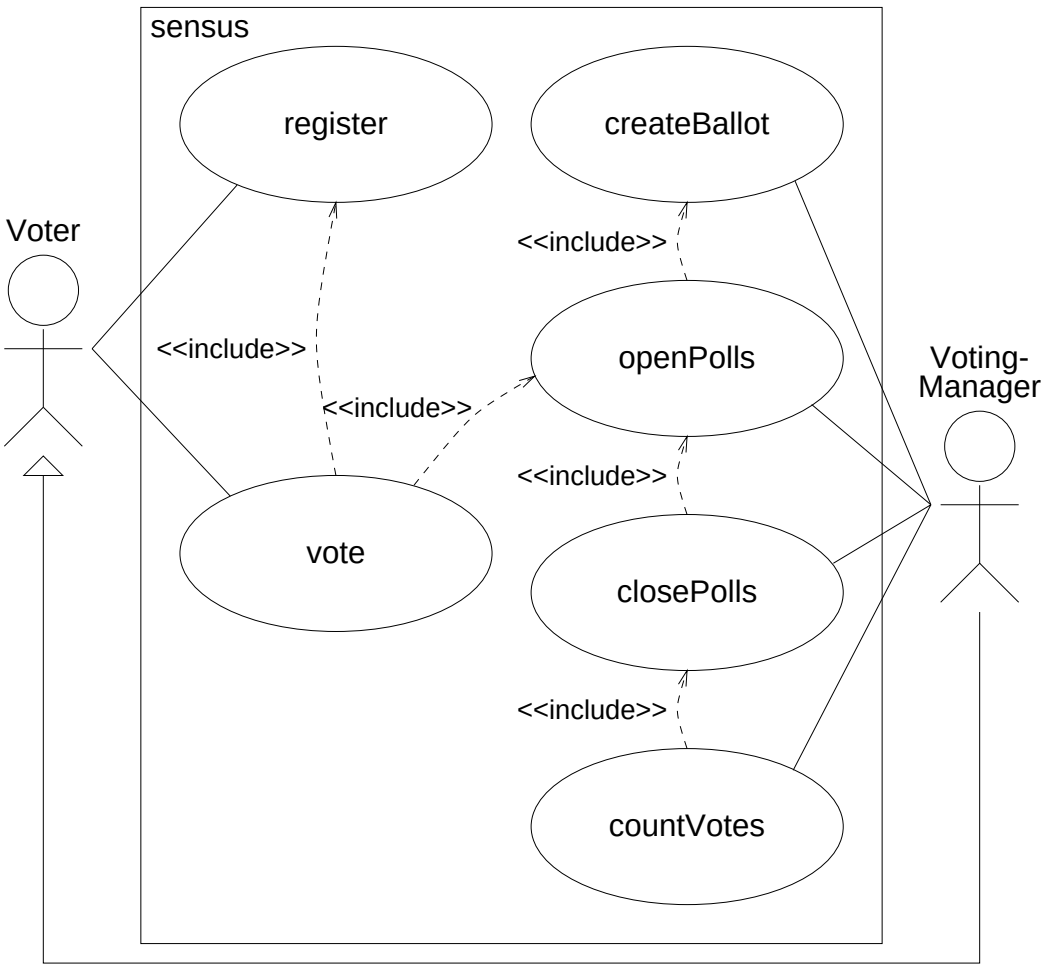
Make a list of questions with which to bug and nudge your client.

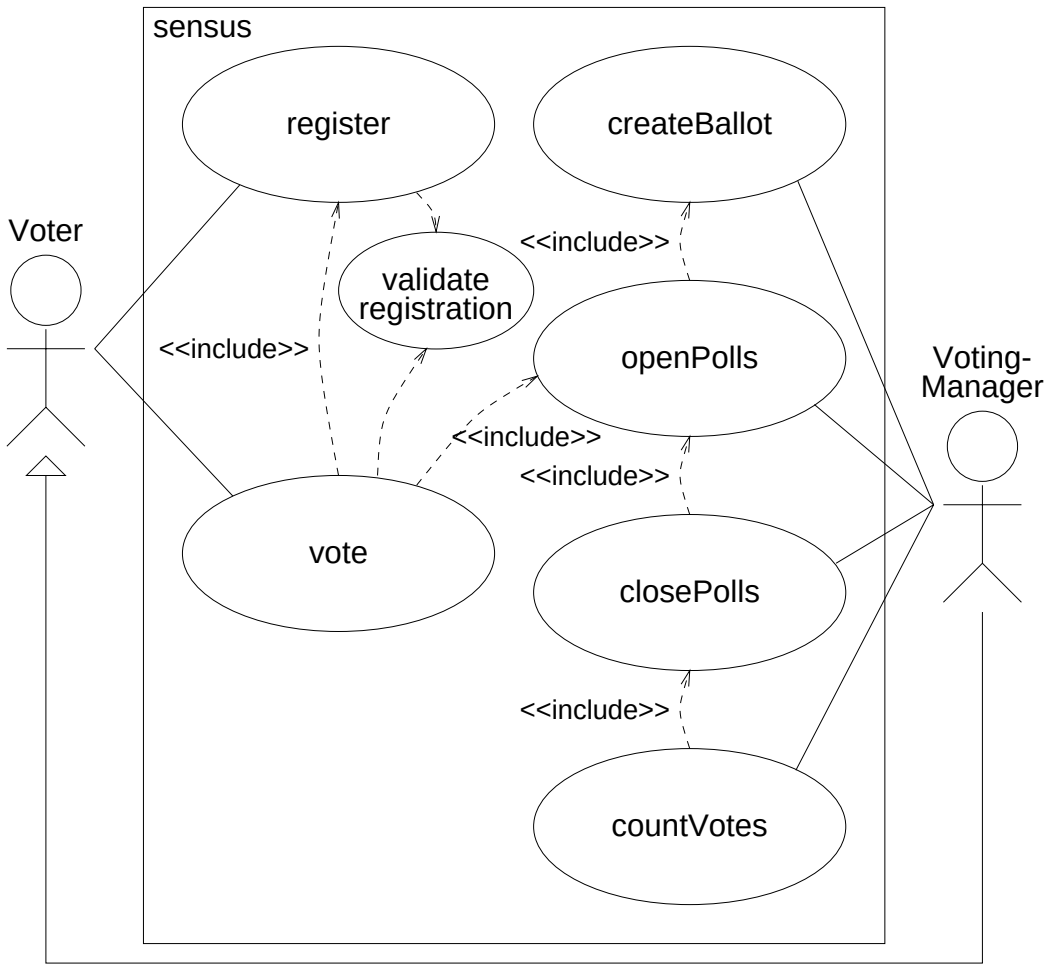
Be a NUDNIK!

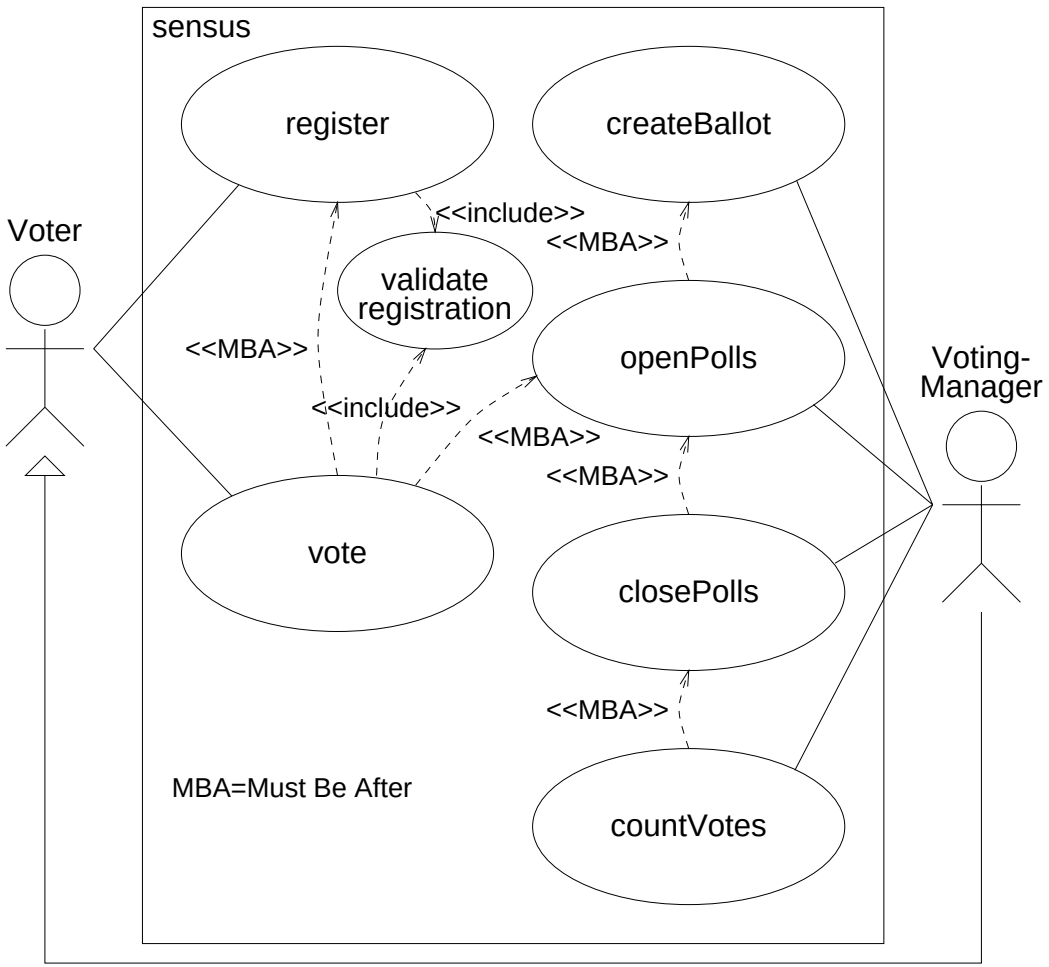
UML Use Case Diagram

The UML use case diagram notation captures all the use cases and actors of a system and shows which actors use which use cases.

The diagram shows also relations between the actors and relations between the use cases.


sensus
register
createBallot
Voter
<<include>>
<<include>>
openPolls
Voting-
Manager
<<include>>
<<include>>
vote
closePolls
<<include>>
countVotes


sensus
register
createBallot
validate registration
<<include>>
<<include>>
openPolls
<<include>>
<<include>>
vote
closePolls
<<include>>
countVotes
Voter
Voting-Manager


sensus
register
createBallot
validate registration
openPolls
vote
closePolls
countVotes
Voter
Voting-Manager
<<include>>
<<MBA>>
<<MBA>>
<<include>>
<<MBA>>
<<MBA>>
<<MBA>>
MBA=Must Be After

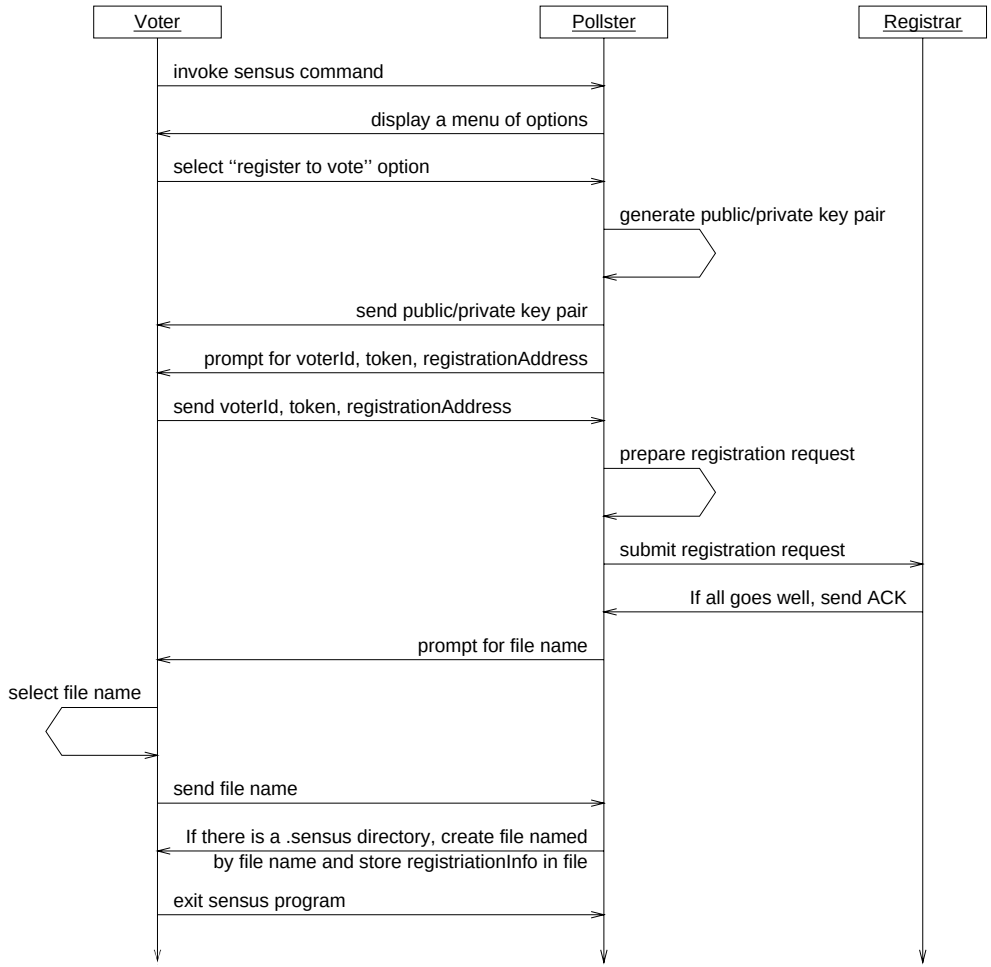
Sequence Diagrams

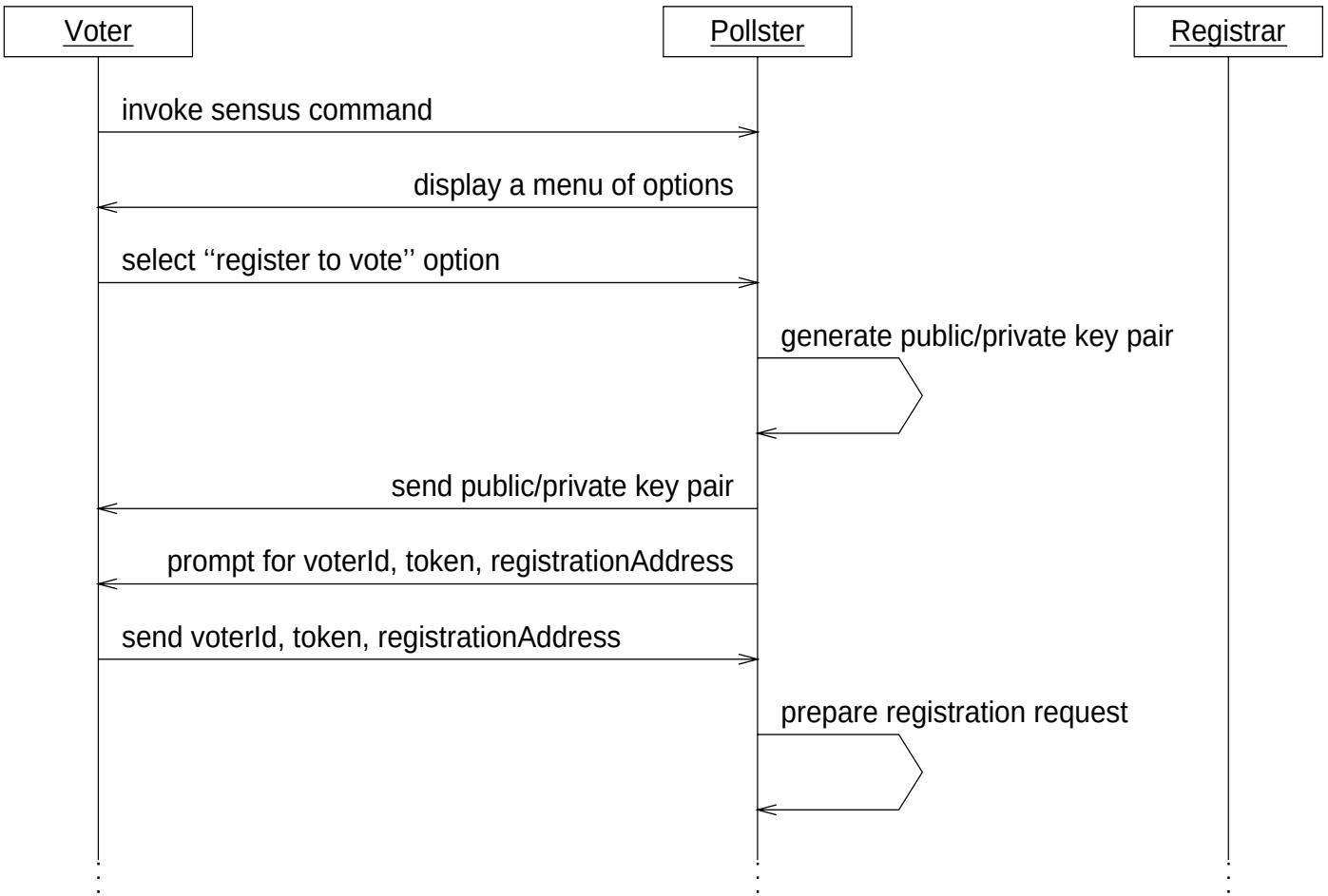
UML has a language for describing scenarios, that of the sequence diagram.

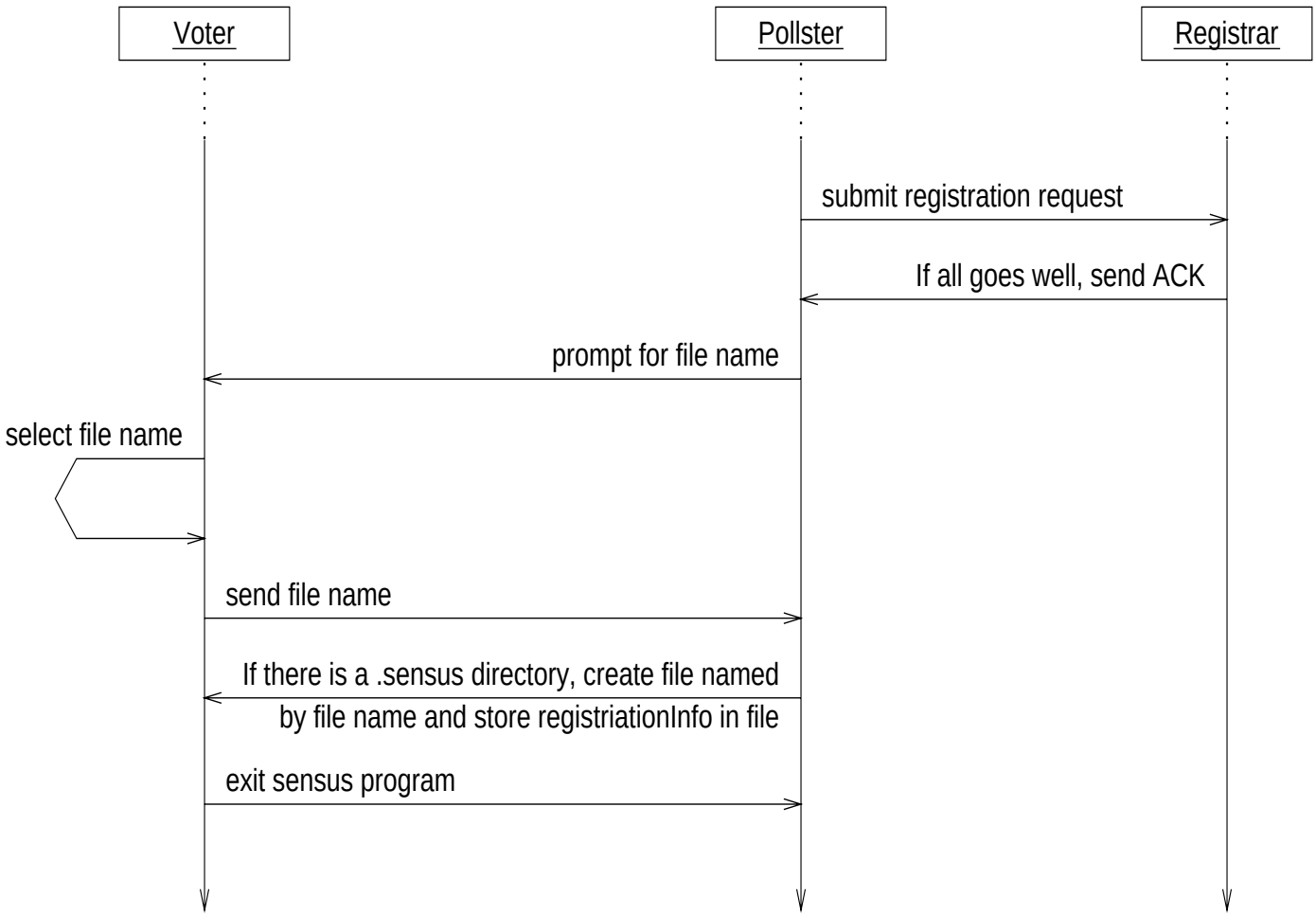
We now look at sequence diagrams for the typical scenarios for the two use cases shown before.

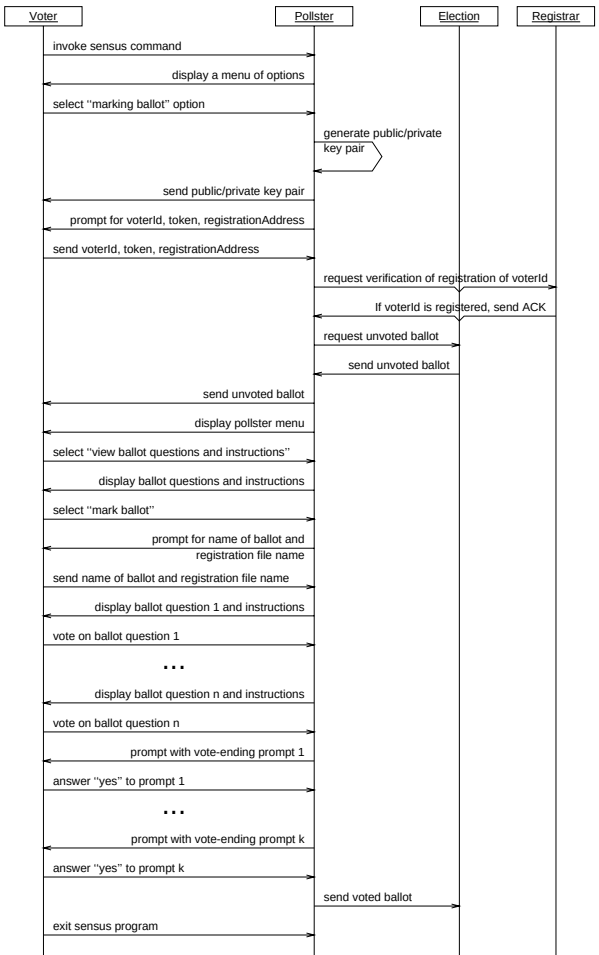
Each is too big to fit on a page. So we first get a bird's eye view and then we look at digestable chunks.

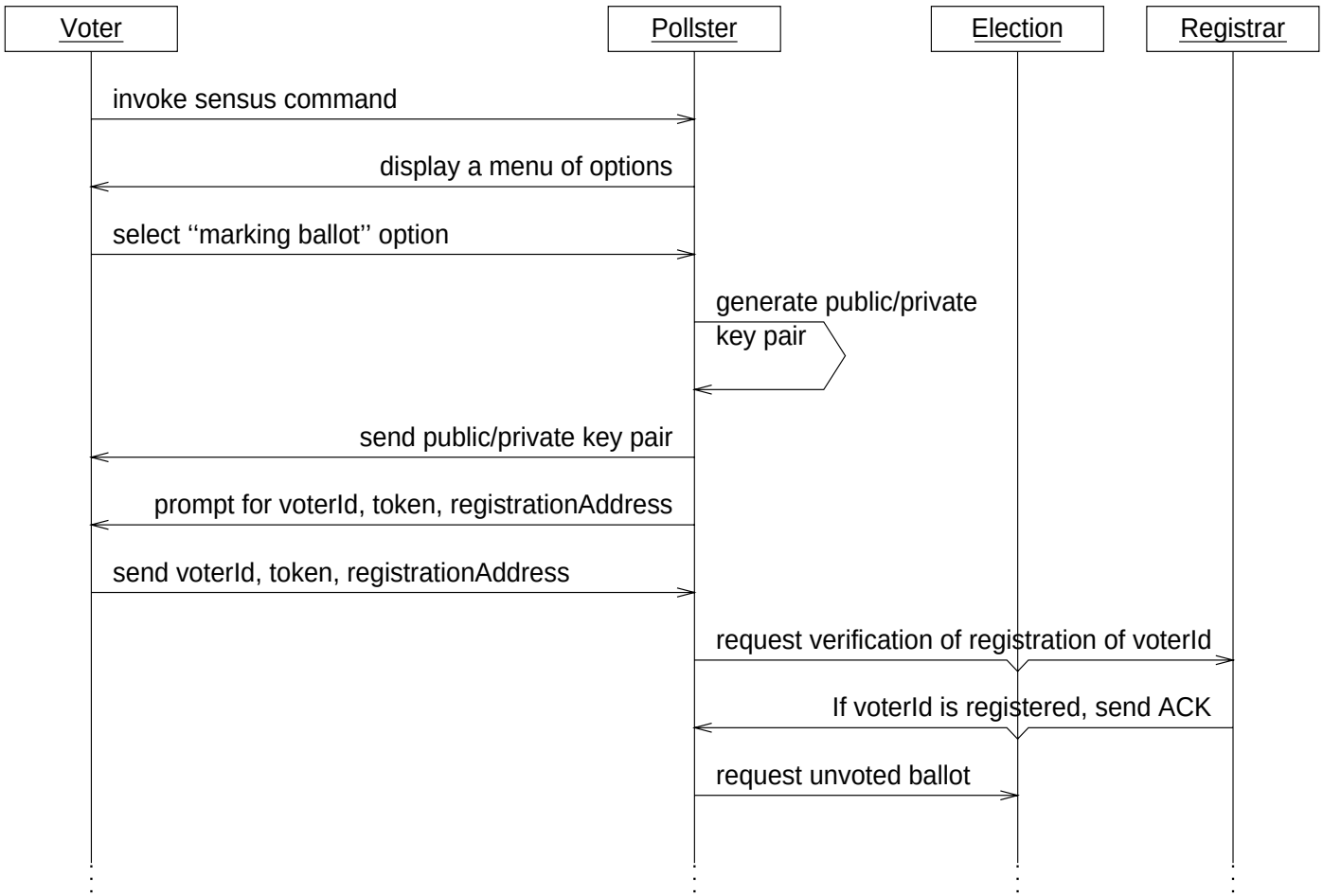
First comes the typical scenario of the “Registering to Vote” use case and then comes the typical scenario of the “Marking your Ballot” use case.

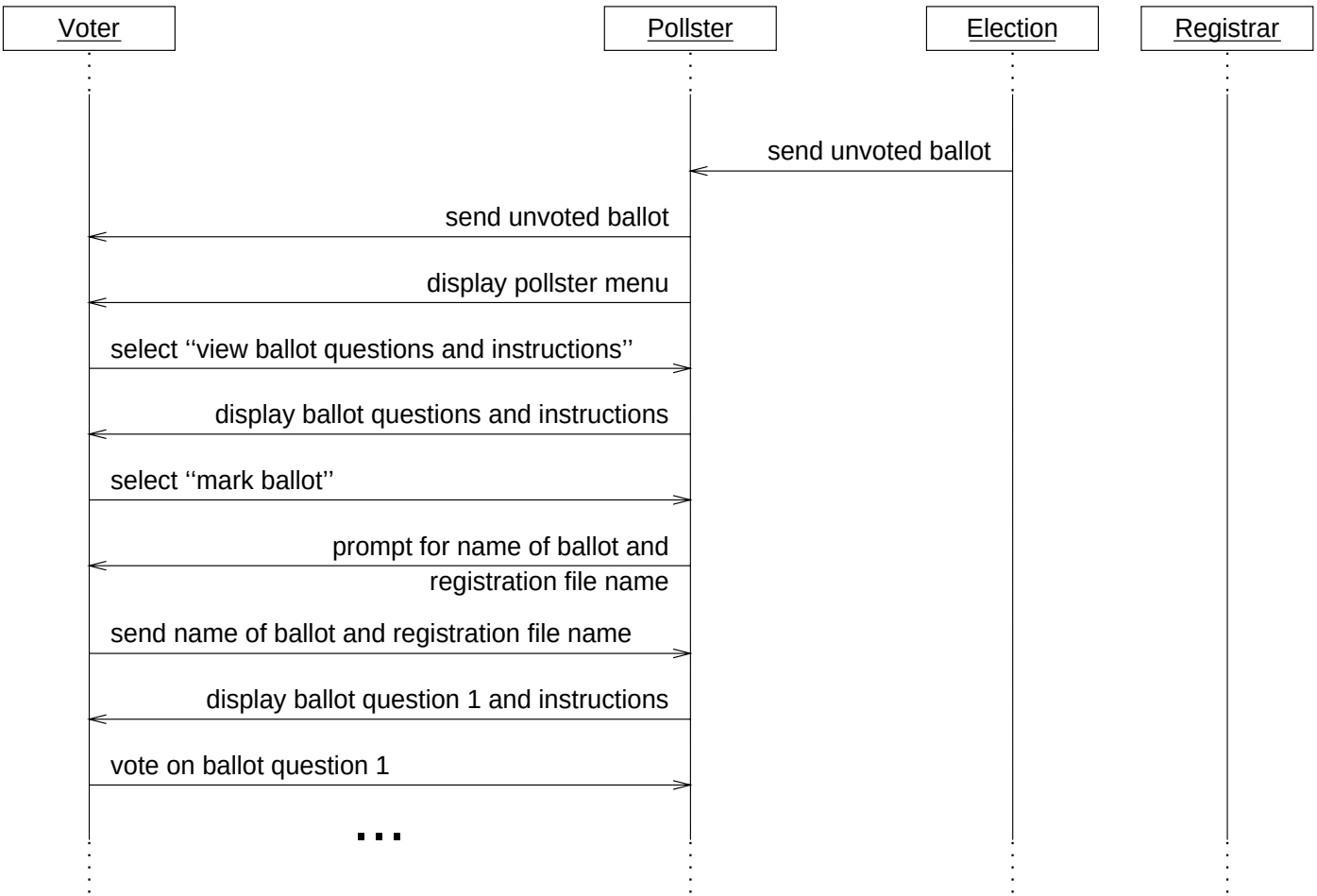

Voter
Pollster
Registrar
invoke sensus command
display a menu of options
select ‘‘register to vote’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
prepare registration request
submit registration request
If all goes well, send ACK
prompt for file name
select file name
send file name
If there is a .sensus directory, create file named
by file name and store registriationInfo in file
exit sensus program

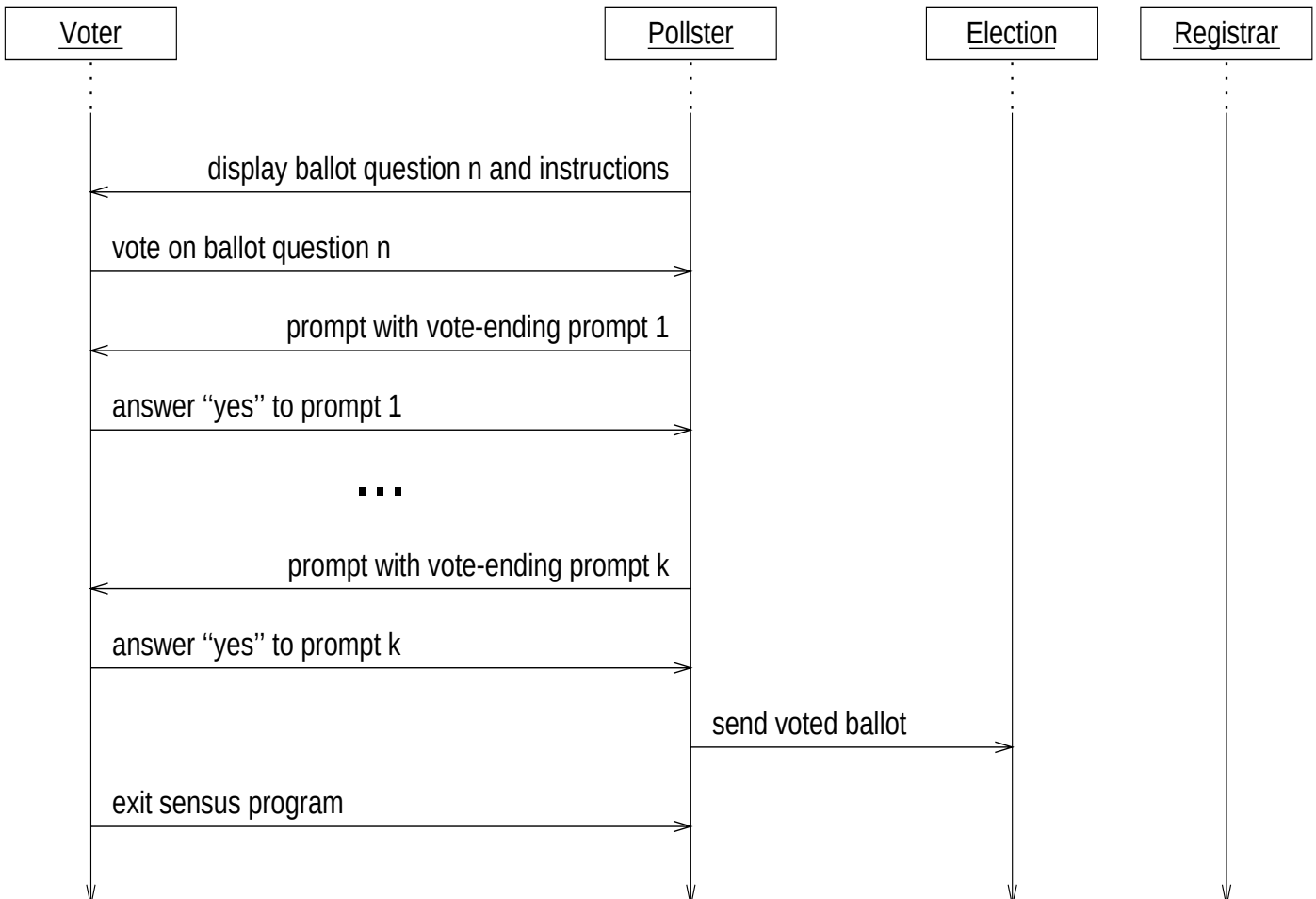

Voter
Pollster
Registrar
invoke sensus command
display a menu of options
select ‘‘register to vote’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
prepare registration request


Voter
Pollster
Registrar
submit registration request
If all goes well, send ACK
prompt for file name
select file name
send file name
If there is a .sensus directory, create file named
by file name and store registriationInfo in file
exit sensus program


Voter
Pollster
Election
Registrar
invoke sensus command
display a menu of options
select ''marking ballot'' option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
request verification of registration of voterId
If voterId is registered, send ACK
request unvoted ballot
send unvoted ballot
send unvoted ballot
display pollster menu
select ''view ballot questions and instructions''
display ballot questions and instructions
select ''mark ballot''
prompt for name of ballot and registration file name
send name of ballot and registration file name
display ballot question 1 and instructions
vote on ballot question 1
. . .
display ballot question n and instructions
vote on ballot question n
prompt with vote-ending prompt 1
answer ''yes'' to prompt 1
. . .
prompt with vote-ending prompt k
answer ''yes'' to prompt k
send voted ballot
exit sensus program


Voter
Pollster
Election
Registrar
invoke sensus command
display a menu of options
select ‘‘marking ballot’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
request verification of registration of voterId
If voterId is registered, send ACK
request unvoted ballot


Voter
Pollster
Election
Registrar
send unvoted ballot
send unvoted ballot
display pollster menu
select ‘‘view ballot questions and instructions’’
display ballot questions and instructions
select ‘‘mark ballot’’
prompt for name of ballot and registration file name
send name of ballot and registration file name
display ballot question 1 and instructions
vote on ballot question 1
...


Voter
Pollster
Election
Registrar
display ballot question n and instructions
vote on ballot question n
prompt with vote-ending prompt 1
answer ‘‘yes’’ to prompt 1
. . .
prompt with vote-ending prompt k
answer ‘‘yes’’ to prompt k
send voted ballot
exit sensus program

Review of Ss & UCs

We have seen how Ss & UCs can help in identifying requirements, i.e., in requirements elicitation.

They give something concrete that the users and clients can latch on to to see how the proposed system will interact with them.

Ss & UCs Useful Otherwise

Ss & UCs are useful for other activities as we shall see in future lectures:

- **as a basis for user validation of requirements understanding**
- **as a basis of an active review of specifications**
- **as a basis of a quick and dirty prototype**
- **as a basis of test cases for the eventual implementation**

Wow!!

Pluses for Ss & UCs

According to Jo Atlee:

- ⊕ **Ss & UCs are simple, easy to create.**
- ⊕ **All clients understand S&Uc. Often the Ss & UCs are the only things the clients understand of the requirements document.**

Pluses, Cont'd

- **Ss & UCs usually reflect the user's essential requirements. You will want to keep the use cases in mind as the specification document is elaborated. It is easy to get carried away and populate specification with bell and whistle functionality. You want to be sure that the core functionality, captured by the use cases, is not lost.**

Pluses, Cont'd

- **Ss & UCs separate normal behaviour from exceptional behavior, making it easier to scrutinize various alternatives, since each can be considered separately.**

Negatives for Ss & UCs

According to Jo Atlee:

- ⊖ **Scenarios don’t scale well in size or complexity: As requirements and scenarios grow in size (usually exponentially), it gets harder and harder to coherently organize them. Abstracting scenarios into use cases helps a bit.**

UC “Registering to Vote”

Voter	Pollster	Registrar
1. Before registering to vote, a voter must have obtained a voter [identification number, token, and registration address] from the election administrators.		
2. Voter invokes the *sensus* command, to run the Pollster.		
	3. Pollster displays a menu of options.	
4. Voter selects the ‘‘register to vote’’ option.		
	5. Pollster generates and sends a public/private key pair for Voter.	
	6. Pollster prompts voter for his/her [identification number, token, and registration address].	

Voter	Pollster	Registrar
7. Voter sends his/her [identification number, token, and registration address].		
	8. Pollster prepares registration request on Voter's behalf.	
	9. Pollster submits registration request to registrar	
		10. If all goes well, Registrar sends acknowledgement to Pollster within a few seconds.
	11. Pollster prompts Voter for a file name for saving his/her registration information.	
12. Voter selects a name he/she will remember; this name must be unique for the registration authority.		

Voter	Pollster	Registrar
13. Voter sends selected name to Pollster.		
	14. If Voter has a *.sensus* directory, Pollster creates file with selected name in Voter's *.sensus* directory and stores registration information in the file.	
15. Voter exits *sensus* program.		

EXCEPTION for step 10:

Voter	Pollster	Registrar
		10. If all does not go well, ...

EXCEPTION for step 14:

Voter	Pollster	Registrar
	14. If Voter does not have a *.sensus* directory, Pollster creates one and then Pollster creates file with selected name in Voter's *.sensus* directory and stores registration information in the file.	

ALTERNATIVES for step 7:

Voter	Pollster	Registrar
7. Voter sends his/her [token, identification number, and registration address].		

... one for each permutation of the data...

UC “Marking your Ballot”

Voter	Pollster	Election	Registrar
1. Voter invokes the *sensus* command, to run the Pollster.			
	2. Pollster displays a menu of options		
3. Voter slects the ‘‘marking ballot’’ option.			
	4. Pollster generates and sends a public/private key pair for Voter.		
	5. Pollster prompts voter for his/her [identification number, token, and registration address].		

Voter	Pollster	Election	Registrar
6. Voter sends his/her [identification number, token, and registration address].			
	7. Pollster requests verification of voterId from Registrar.		
			8. If voterId is registered, Registrar sends Pollster acknowledgement.
	9. Pollster requests an unvoted ballot from Election authority and sends it to Voter.		
		10. Election authority sends an unvoted ballot to Pollster	
	11. Pollster sends unvoted ballot to Voter.		

Voter	Pollster	Election	Registrar
	12. Pollster displays pollster menu.		
13. Voter selects ‘‘view ballot questions and instructions’’ from the pollster menu.			
	14. Pollster displays ballot questions and instructions.		
15. Voter selects ‘‘mark ballot’’ from the pollster menu.			
	16. Pollster prompts for the name of the ballot and Voter’s registration file name.		
17. Voter sends the name of the ballot and Voter’s registration file name.			

Voter	Pollster	Election	Registrar
	18. Pollster displays ballot question 1 and its voting instructions.		
19. Voter votes on ballot question 1.			
...			
	20. Pollster displays ballot question *m* and its voting instructions.		
21. Voter votes on ballot question *m*.			
	22. Pollster prompts Voter with vote-ending prompt 1.		
23. Voter answers “yes” to prompt 1.			
...			

Voter	Pollster	Election	Registrar
	24. Pollster prompts Voter with vote-ending prompt *k*.		
25. Voter answers “yes” to prompt *k*.			
	26. Pollster sends completely voted ballot to Election Authority.		
27. Voter exits *sensus* program.			

ALTERNATIVE for steps 13–14 for Voter that does not want to review the ballot before marking it:

Voter	Pollster	Election	Registrar
After step 12 comes step 15.			

ALTERNATIVE for steps 18–21 for Voter that quits after some number of ballot questions and goes back to remark the entire ballot, because he or she has changed his or her mind or has made a mistake:

Voter	Pollster	Election	Registrar
After last step among 18–21, go to step 15.			

EXCEPTIONS for steps 22–25:

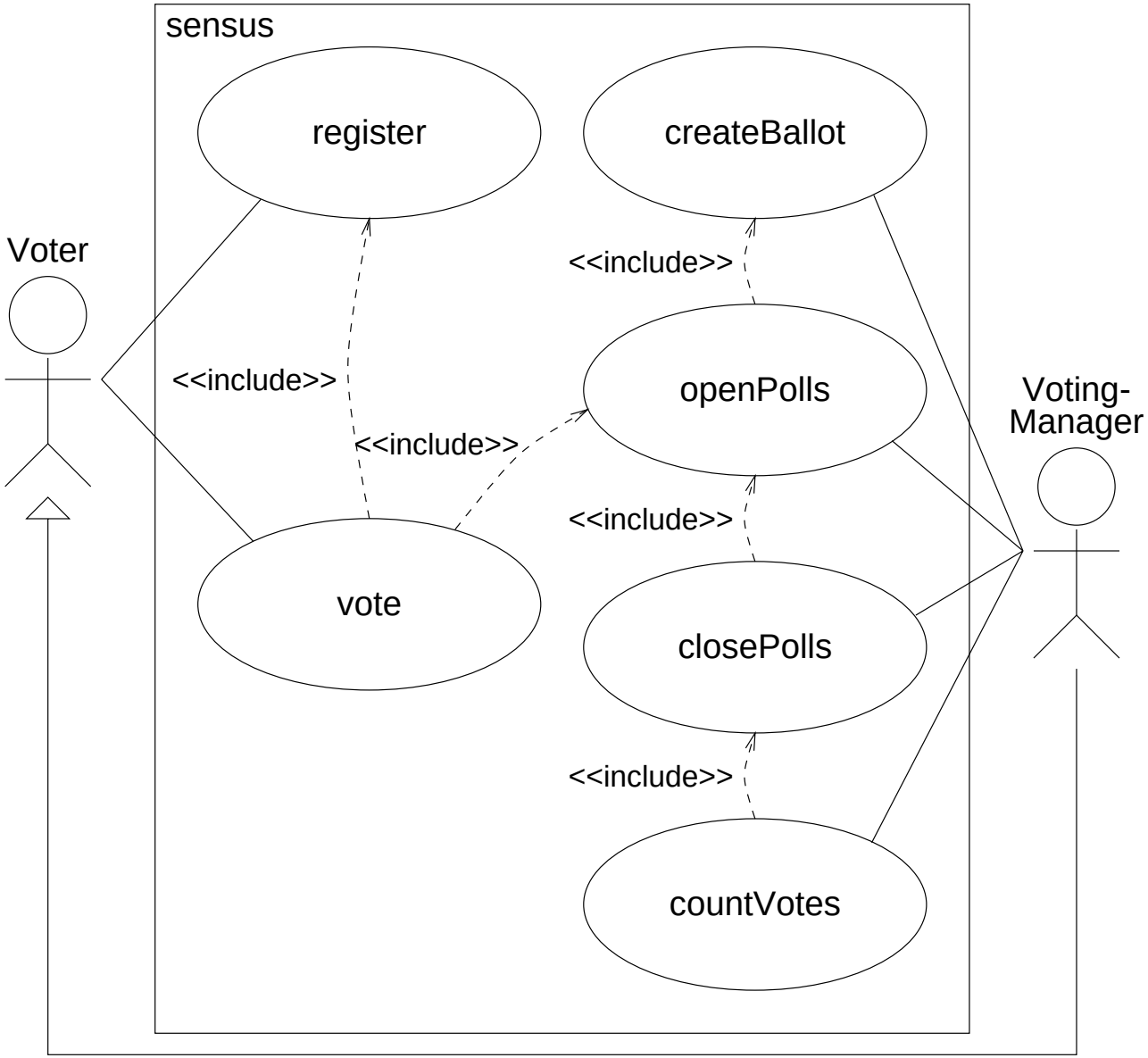
Voter	Pollster	Election	Registrar
Voter answers “no” to a vote-ending prompt *i*<*k* and exits the *sensus* program, able to continue from the last completed step.			

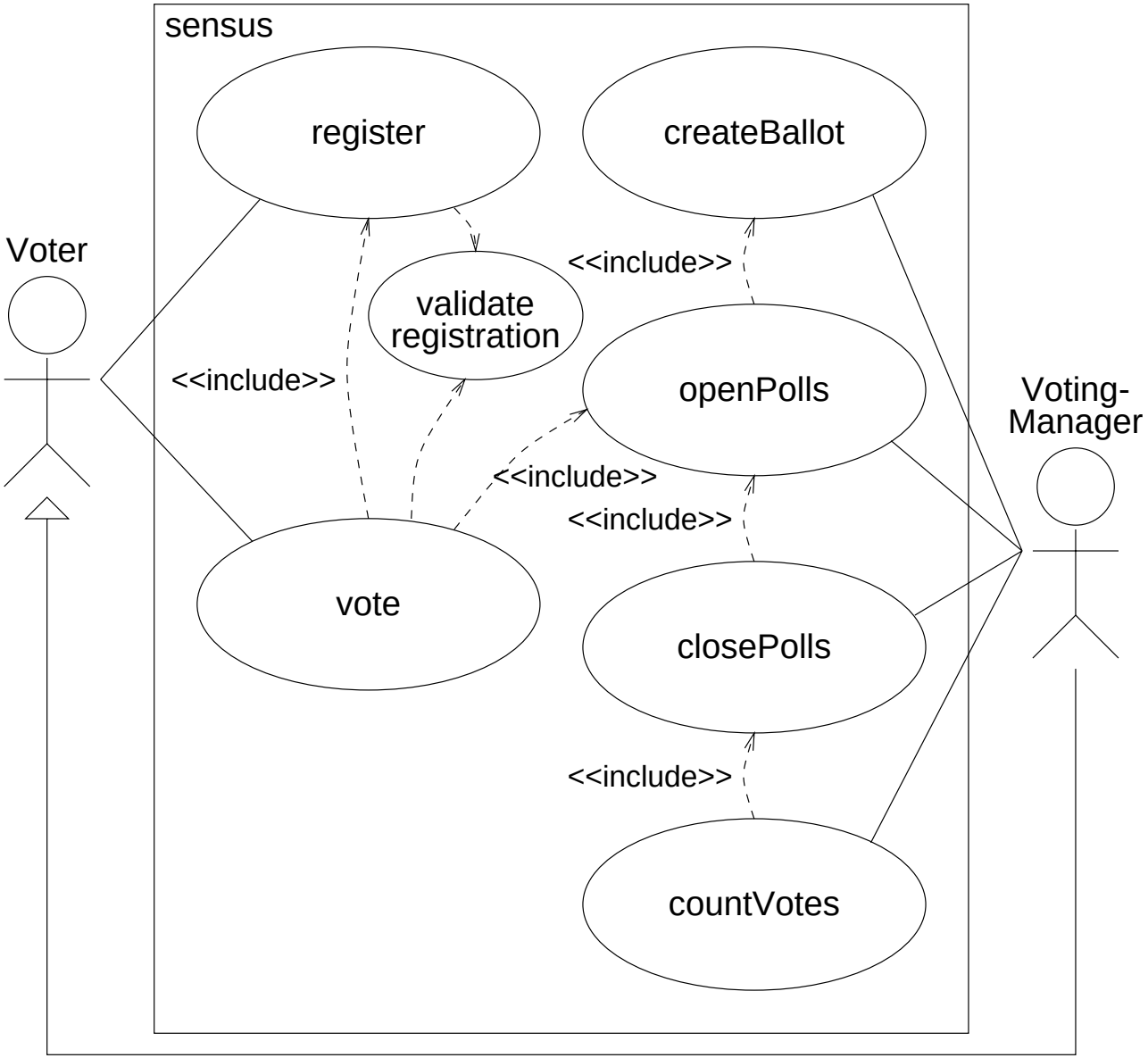
EXCEPTION for step 8:

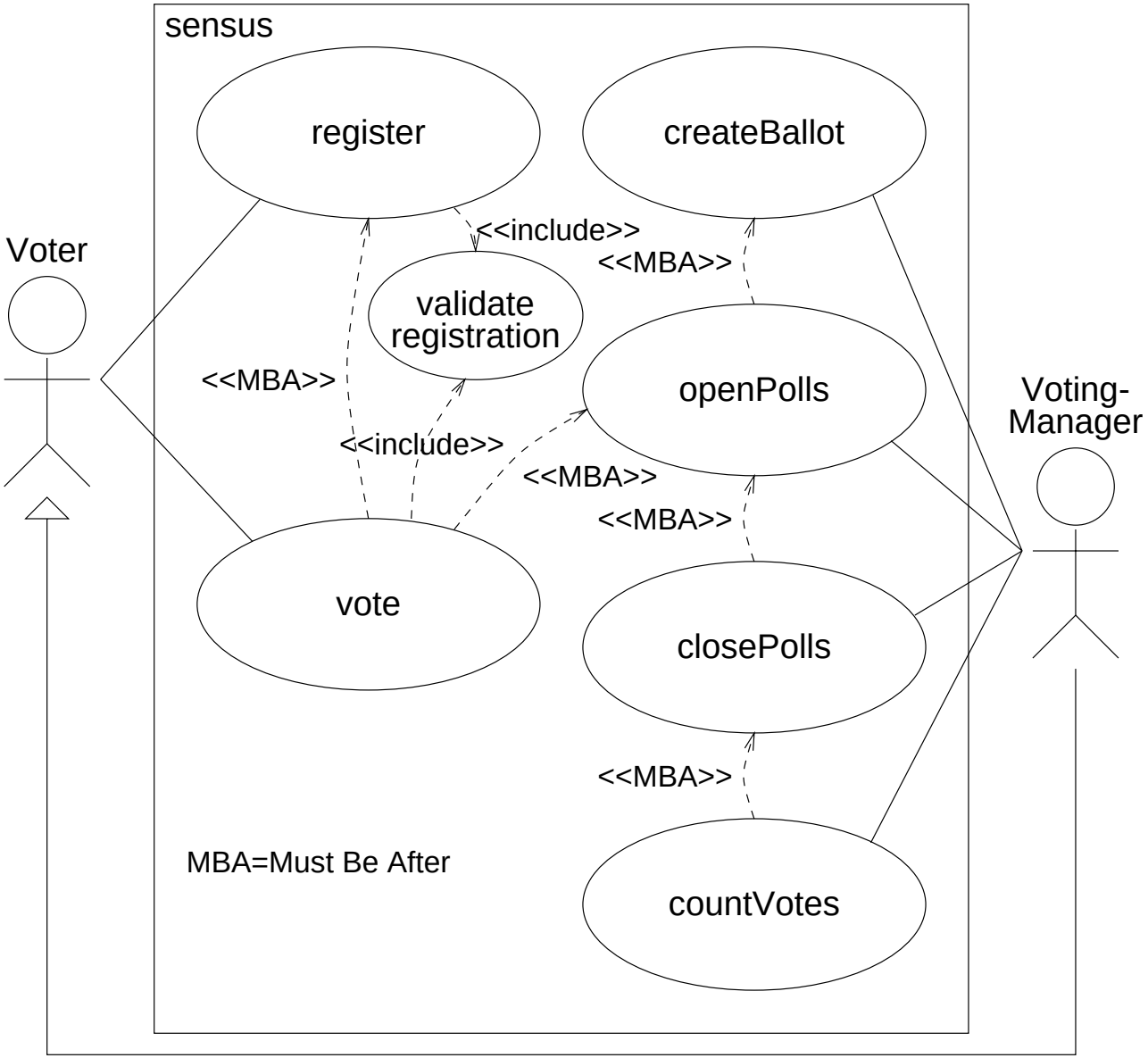
Voter	Pollster	Election	Registrar
voterId is not registered ...			

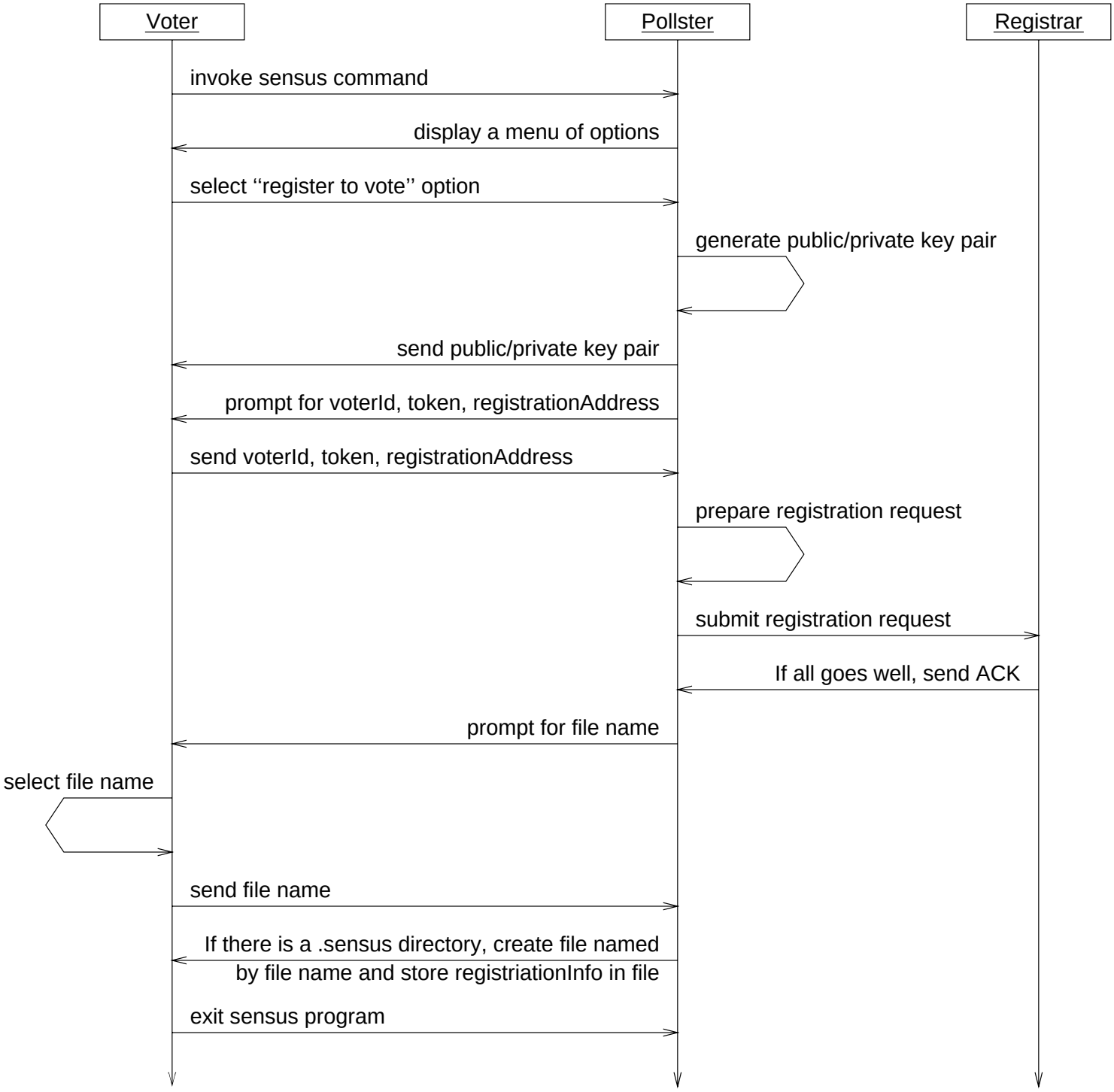
ALTERNATIVE for any Voter step:

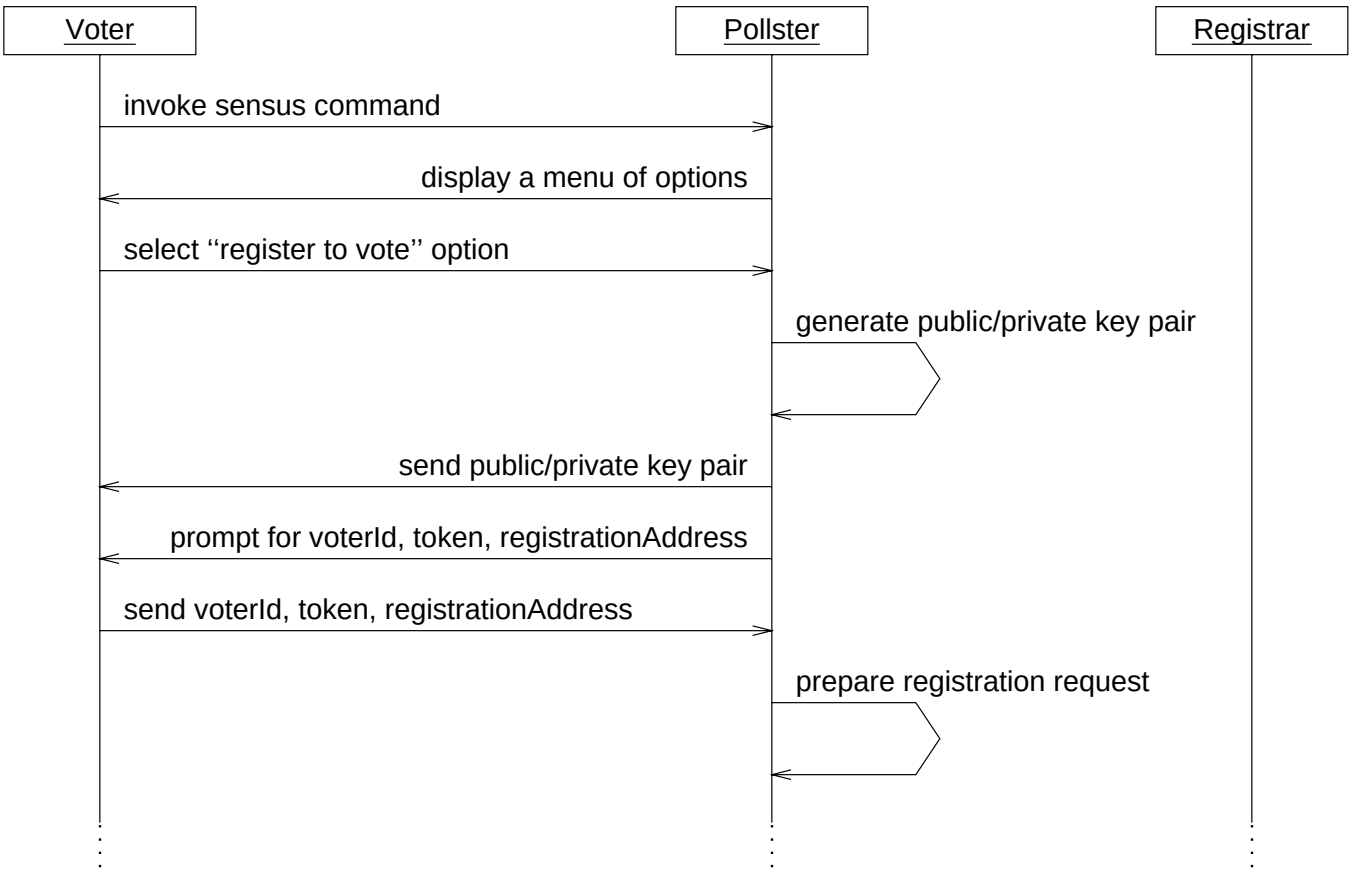
Voter	Pollster	Election	Registrar
Voter exits *sensus* program, able to continue from the last completed step.			

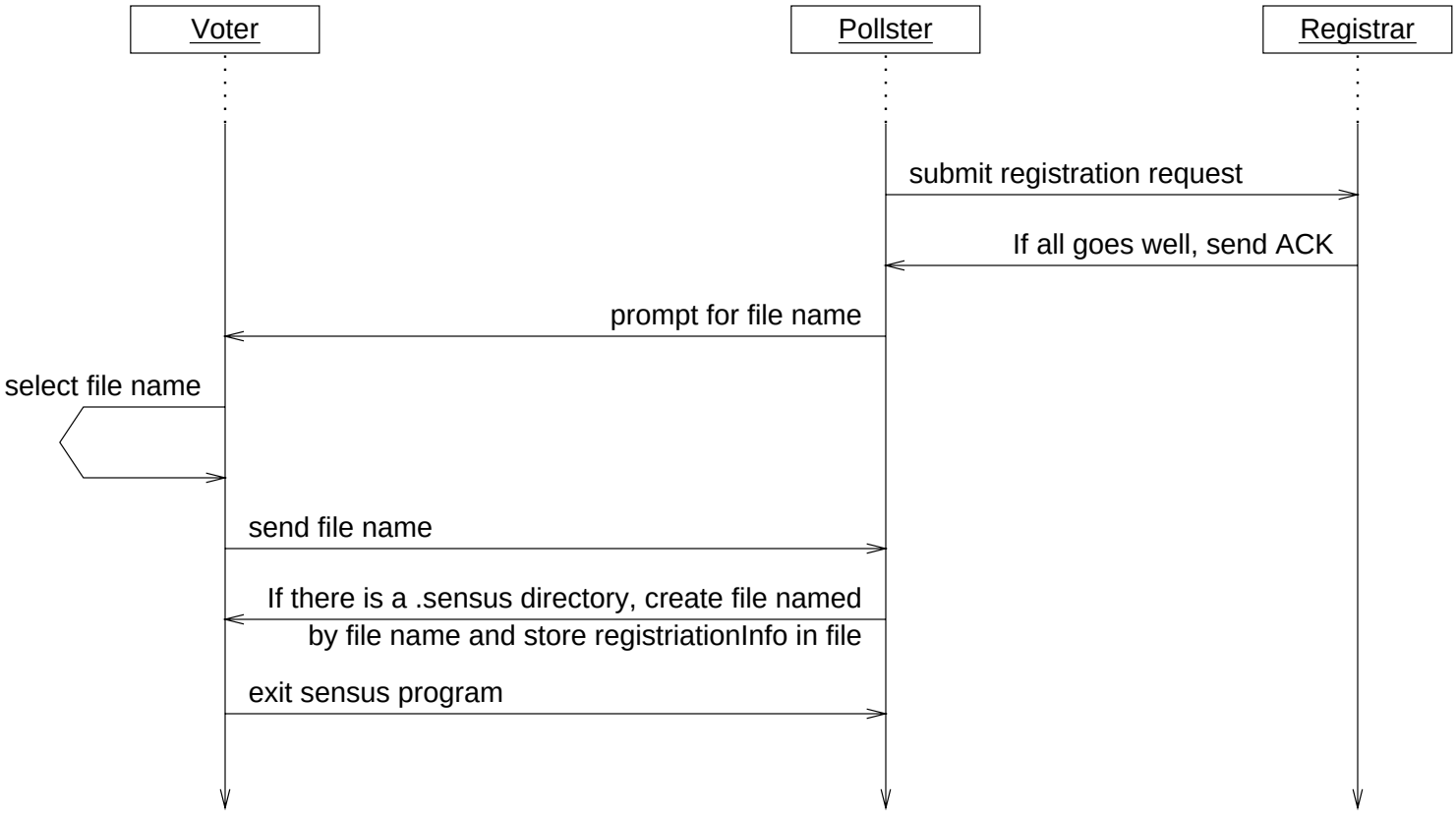

sensus
register
createBallot
Voter
<<include>>
openPolls
Voting-
Manager
<<include>>
<<include>>
vote
<<include>>
closePolls
<<include>>
countVotes

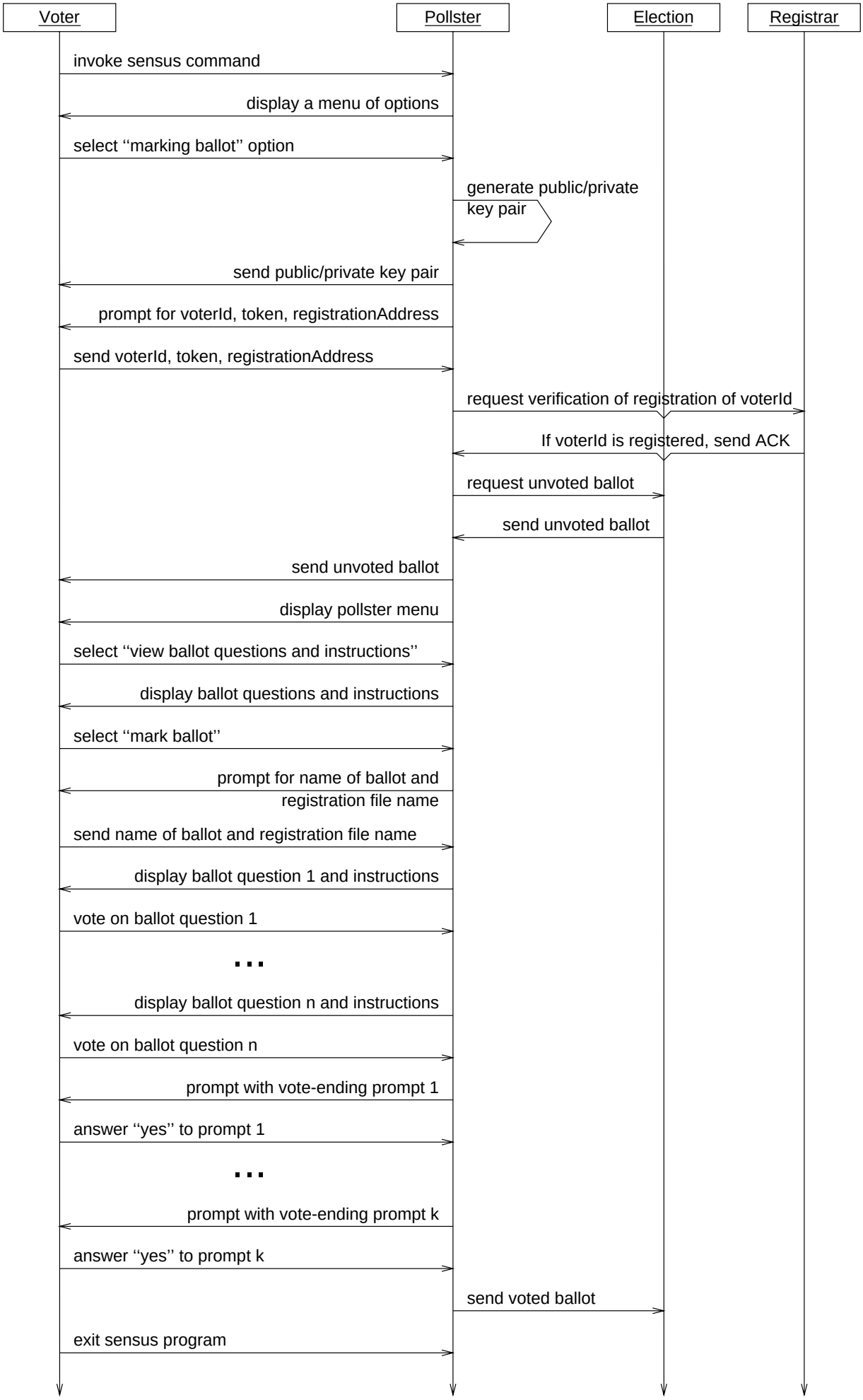

sensus
register
createBallot
validate registration
<<include>>
Voter
<<include>>
openPolls
Voting-Manager
<<include>>
<<include>>
vote
closePolls
<<include>>
countVotes

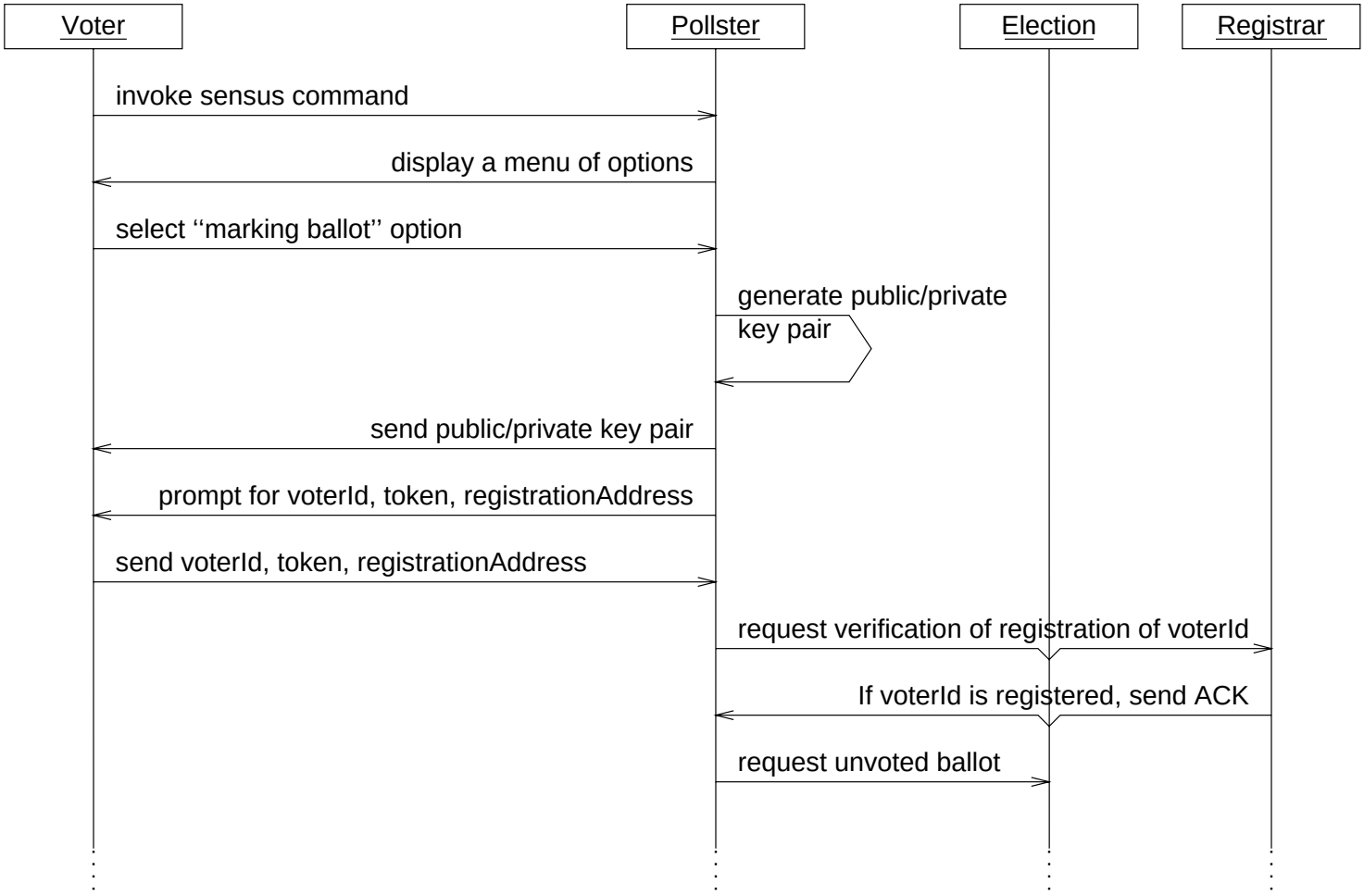

sensus
register
createBallot
<<include>>
<<MBA>>
validate registration
Voter
<<MBA>>
openPolls
Voting-Manager
<<include>>
<<MBA>>
<<MBA>>
vote
closePolls
<<MBA>>
MBA=Must Be After
countVotes

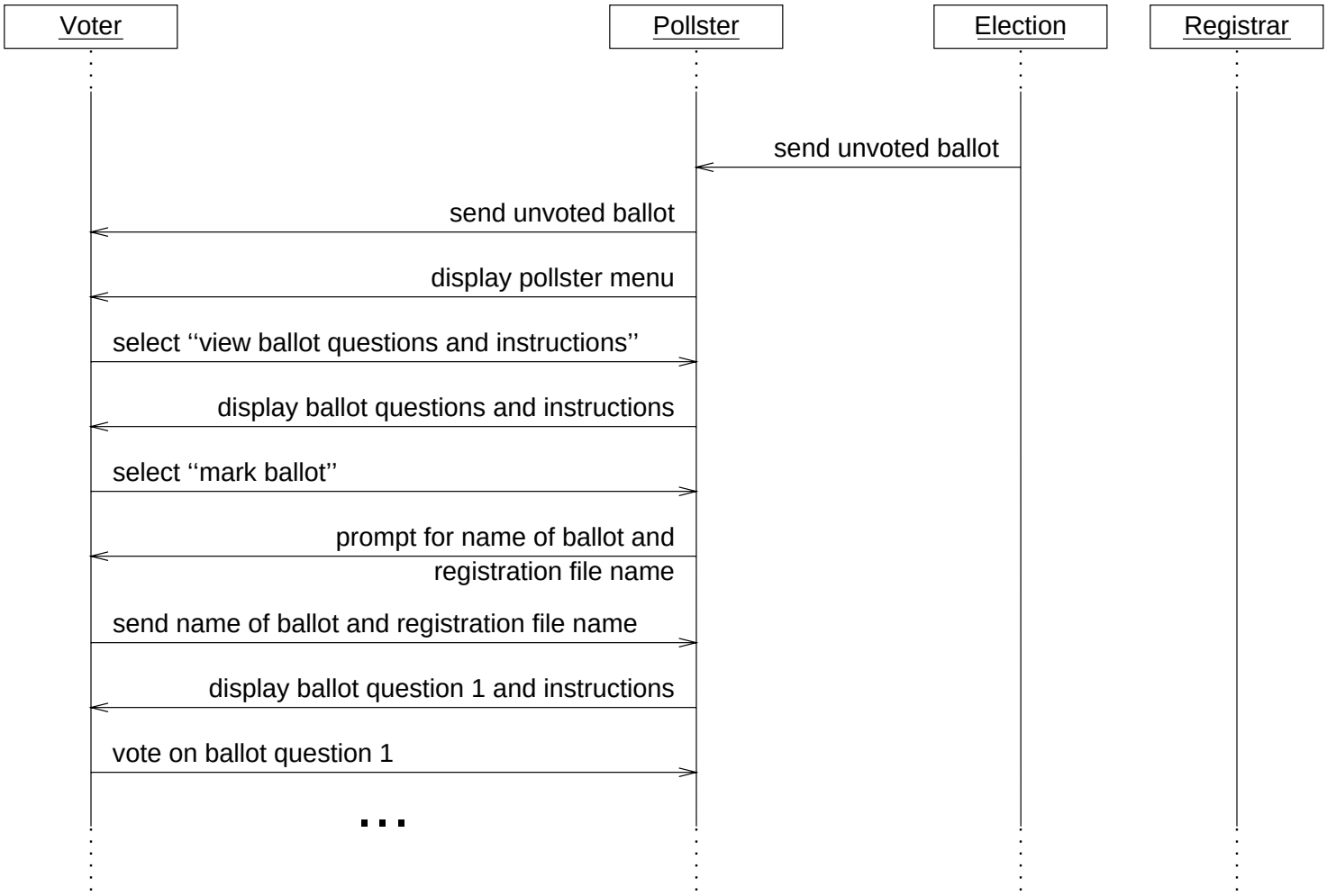

Voter
Pollster
Registrar
invoke sensus command
display a menu of options
select ‘‘register to vote’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
prepare registration request
submit registration request
If all goes well, send ACK
prompt for file name
select file name
send file name
If there is a .sensus directory, create file named
by file name and store registriationInfo in file
exit sensus program

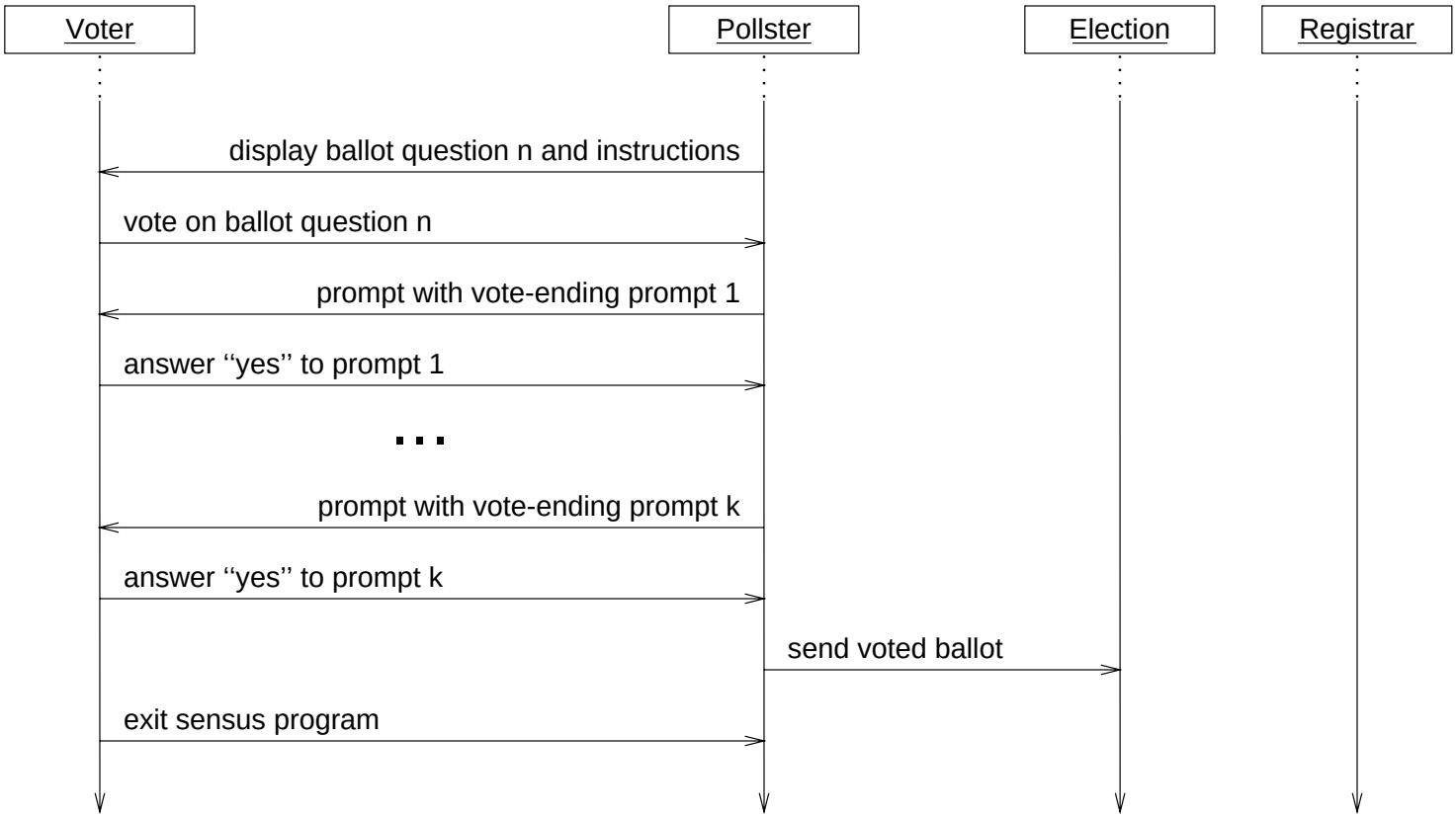

Voter
Pollster
Registrar
invoke sensus command
display a menu of options
select ‘‘register to vote’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
prepare registration request


Voter
Pollster
Registrar
submit registration request
If all goes well, send ACK
prompt for file name
select file name
send file name
If there is a .sensus directory, create file named
by file name and store registriationInfo in file
exit sensus program


Voter
Pollster
Election
Registrar
invoke sensus command
display a menu of options
select ‘‘marking ballot’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
request verification of registration of voterId
If voterId is registered, send ACK
request unvoted ballot
send unvoted ballot
send unvoted ballot
display pollster menu
select ‘‘view ballot questions and instructions’’
display ballot questions and instructions
select ‘‘mark ballot’’
prompt for name of ballot and registration file name
send name of ballot and registration file name
display ballot question 1 and instructions
vote on ballot question 1
. . .
display ballot question n and instructions
vote on ballot question n
prompt with vote-ending prompt 1
answer ‘‘yes’’ to prompt 1
. . .
prompt with vote-ending prompt k
answer ‘‘yes’’ to prompt k
send voted ballot
exit sensus program


Voter
Pollster
Election
Registrar
invoke sensus command
display a menu of options
select ‘‘marking ballot’’ option
generate public/private key pair
send public/private key pair
prompt for voterId, token, registrationAddress
send voterId, token, registrationAddress
request verification of registration of voterId
If voterId is registered, send ACK
request unvoted ballot


Voter
Pollster
Election
Registrar
send unvoted ballot
send unvoted ballot
display pollster menu
select ‘‘view ballot questions and instructions’’
display ballot questions and instructions
select ‘‘mark ballot’’
prompt for name of ballot and registration file name
send name of ballot and registration file name
display ballot question 1 and instructions
vote on ballot question 1
. . .


Voter
Pollster
Election
Registrar
display ballot question n and instructions
vote on ballot question n
prompt with vote-ending prompt 1
answer ‘‘yes’’ to prompt 1
. . .
prompt with vote-ending prompt k
answer ‘‘yes’’ to prompt k
send voted ballot
exit sensus program